Logic16

Time Tag Unit

User Manual

Version 3.0

UQDevices

Logic16 Time Tag Unit
User Manual

Version 3.0

UQDevices

Universal Quantum Devices

295 Hagey Boulevard, 1st Floor, West Entrance, Waterloo, ON, Canada N2L 6R5

Phone: +1 (519) 778-9146 | Email: info@uqdevices.com | Web: www.uqdevices.com

In collaboration with:

dotfast-consulting.at

Table of contents

Section 1

Introduction

Logic16 is a versatile time tagger and logic correlation unit made especially for analyzing photon detection signals in multi-photon correlation counting and time-resolved signal analysis.

Logic16 operates in two distinct modes for measurement flexibility: Logic Mode and Time Tagger Mode.

Logic Mode performs on-board coincidence (pattern) counting and can internally process *all* possible patterns for the 16 inputs, simultaneously. The patterns to be counted are defined by indicating for each input whether or not an active signal edge should appear, or if the input should be ignored. In Logic Mode, the unit can also output TTL pulses triggered by detections of user-selected coincidence patterns. Output channels are distinct from input channels. Coincidence counting can be performed alongside the use of the pattern-triggered outputs. Since all time-correlation processing is performed internally, it can be used even with small, low-power computers.

Time Tagger Mode *time tags* photon events – in other words, it records the input channel and event time when signals are detected. In Time Tagger Mode it is possible to perform:

continuous time tagging	- time tag each event detected
gated time tagging	- condition time tagging to occur only when signals appear on a specific input
time tag filter	- time tag events only when they appear in groups - use for time-of-flight[1] measurements with respect to a fast pump signal[2] - use to reduce data file size for long[3] measurements

By employing the time tag filter it is possible for users to measure with sustained[4] data rates up to 190 MHz. However, keep in mind that time tags can be processed on-board Logic16 faster than they can be sent across the USB interface to the host PC. The time Tag Filter is thus useful for high rate signals when only select events are of interest.

---

[1] Or “time-of-arrival” (TOA)

[2] For example, a 76 MHz Ti-Sapphire laser

[3] For example, on the order of days

[4] In contrast to burst

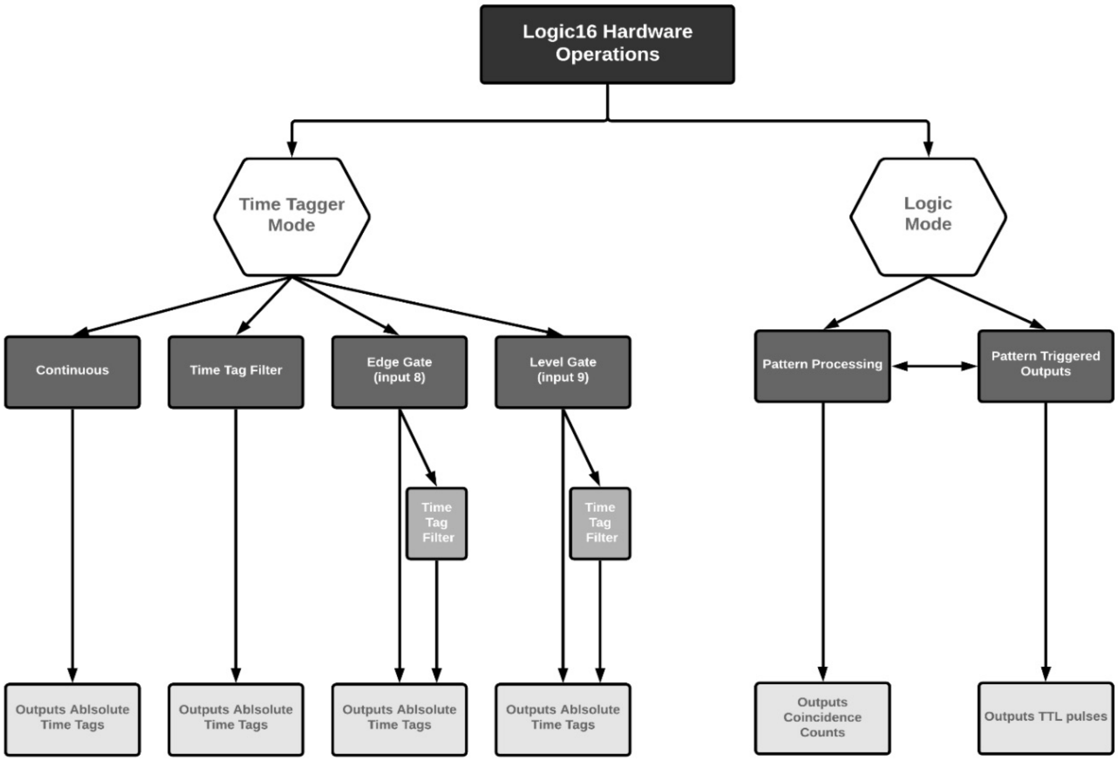

Logic16 Hardware Operations
Time Tagger Mode
Logic Mode
Continuous
Time Tag Filter
Edge Gate (input 8)
Level Gate (input 9)
Pattern Processing
Pattern Triggered Outputs
Time Tag Filter
Time Tag Filter
Outputs Ablsolute Time Tags
Outputs Ablsolute Time Tags
Outputs Ablsolute Time Tags
Outputs Ablsolute Time Tags
Outputs Coincidence Counts
Outputs TTL pulses

Figure 1-I Summary of Logic16 hardware operations

Configurations

Logic16 has two different configurations. Configuration A has 16 input channels with a resolution of 156.25 ps time bin width while Configuration B has 8 operational input channels with a resolution of 78.125 ps time bin width. Unless requested, Configuration A is shipped as a default. It is easy to switch between the two configurations using a user-supplied JTAG cable. See Section 3.3 for details.

1.1 Support

To improve user experience, Universal Quantum Devices (UQD) provides a variety of support to help you get started with Logic16. Included in this manual you will find a quick start guide, software installation tips and detailed information about Logic16 operation. Programming examples are included on the Logic16 CD, which is available at www.uqdevices.com. On the website you will also find video tutorials.

The success of UQD and DotFAST consulting is based on tight interaction with our customers. In the case you experience any problems, need clarifications or simply have an idea for improvement, contact us by phone: +1 (519) 778-9146 or email: info@uqdevices.com.

Disclaimer: This product is subject to constant development and improvement. For latest drivers and documentation please visit the website.

Section 2

Quick Start Guide

This guide outlines basic steps to get started with Logic16. Refer to section 3.1 to identify ports and connectors on the unit. Videos are available on the website for additional getting started support.

i Power the device

Connect Logic16 to a DC power supply (12 V @ ≥1.5 A) or a NIM crate (12 V @ 1.0 A and -12V @0.4 A). The "Power" LED should turn blue.

ii Set up a connection

Connect Logic16 to your computer using the USB 2 cable supplied with your device.

iii Establish communication

Ensure software and program libraries are available on your host PC.[5] Open the Time Tag Explorer software and click *connect*. The "USB" LED should turn blue.

This quick start guide references the use of Time Tag Explorer. See section 4 to learn about Logic16 Correlation Viewer *or custom programming.*

iv Connect to a signal

Connect one of the *input* channels to a photon detector using an SMA Cable.

Alternatively, simulate your signal using output 4 - a basic function generator[6].

v Read tags or count patterns

Check the *Read Tags* box on the Time Tag Explorer *Time Tag* Tab and verify that time tag differences appear in the GUI window. OR go to the *Logic* tab, check *Use Logic* box and verify that counts appear in the GUI window under *singles* for the connected input.
If you cannot see these signals, please check that your input thresholds on the *input* tab are set appropriately for your signal.

---

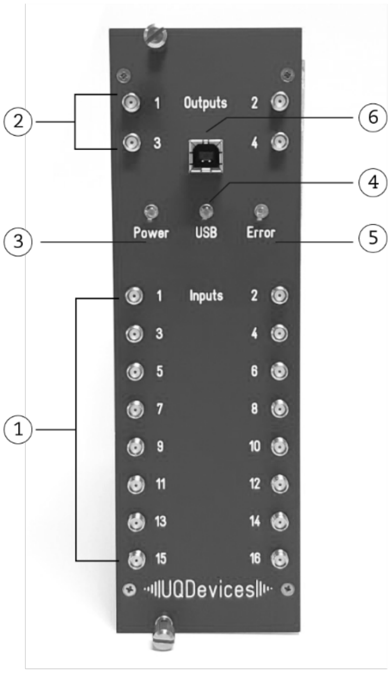
[5] Available at www.uqdevices.com/documentations or on the CD included with Logic16

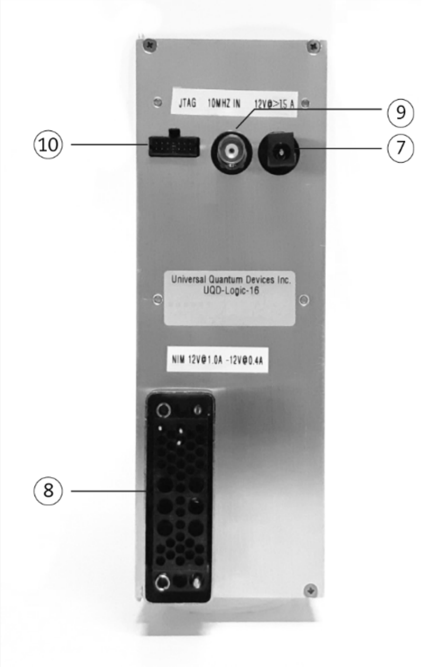
[6] Control the function generator on the "FG" tab of Time Tag Explorer – click the box beside "FG" to turn on

Section 3

Hardware Specifications

3.1 Technical specifications





Front-View

① **Inputs**	SMA, 50Ω termination
② **Outputs**	SMA, 50Ω termination
③ **Power LED**	Blue when unit is connected to a power supply
④ **USB LED**	Blue when unit has established a successful connection to host PC
⑤ **Error LED**	Red when an error occurs during operation of the device
Note: All three LEDs light up briefly upon connection to the power supply. This indicates that the power is on and the unit is booting.	
⑥ **USB port**	Connection for USB 2.0 cable provided.
Note: We recommend that you do not use a USB Hub in order to connect multiple units to your PC. This will limit the data transfer rate of the device.	

Rear-View

⑦ **DC power supply port**	+12 V @ 1.5 A
⑧ **NIM power supply connector**	+12 V @ 1.0 A & -12 V @ 0.4 A
Note: Logic16 can be operated using either a DC power supply OR a NIM Crate.	
⑨ **External Clock Input**	10 MHz external clock BNC input
⑩ **JTAG Connector**	JTAG link cable (e.g. JTAG-cable Xilinx, Xilinx parallel cable, HS2 Digilent)

3.2 Safety of Operation

Ensure the module is operated vertically, and that the air is free to flow through the cooling vents in the top and bottom sides of the unit.
Never operate the module horizontally, or with covered air vents - this could lead to overheating and cause damage.

NIM crate power supply

The modules are designed for operation within a NIM crate that is powered with a properly rated and installed power supply. In particular, in order to ensure safe operation of the logic module and any connected instruments or computers, it is critical that the NIM crate and its power supply are connected to electric ground through the power connector, as required by the local safety standards.

DC power supply

Only use DC regulated power supply which meets the safety standards in your respective country and/or legal jurisdiction.

We also strongly recommend that the DC power supply is grounded when in use.

Disclaimer:

Neither Universal Quantum Devices nor DotFAST Consulting are responsible for any harm or damage caused to or by Logic16 or any connected devices or instruments, in the case that (a) the unit is operated outside of a NIM crate or DC power supply and/or (b) is powered by an incorrectly installed NIM crate or DC power supply.

SPECIFICATIONS

Note: Users can switch between configuration A and B with a firmware update

TIMING	Configuration A	Configuration B
Time bin, digital	156.25 ps	78.125 ps
Time jitter (RMS)	71.6 ps	58.5ps
Dead time	5 ns[†]	5 ns[†]
Time tag filter window	156.25 ps	78.125 ps
Max. input delay	0 to + 40 µs[††] (2^{18} – 1 units)	0 to + 20 µs[††] (2^{18} – 1 units)
Input delay adjustment step	156.25 ps	78.125 ps
Coincidence window	0.156 ns – 2.6 ms (2^{24} -1 units)	0.078ns – 1.3 ms (2^{24} -1 units)
SYSTEM PERFORMANCE		
Time tag rate burst rate	200 Mtags/s for up to 1024 events/channel	
Sustained data rate (time tagger mode)	~ 11 M tags/s (USB 2.0)	
Time tag processing rate, time tag filter mode [†††]	190 M events/s	
Max. on-board coincidence count rate	100 MHz (summed across all channels)	
Input channel settings (for coincidence pattern definition)	Tri-state: Active, Inactive, Ignore	
Maximum number of simultaneously processed patterns	> 43 million (3^{16})	> 6 thousand (3^{8})
INPUTS		
Number of inputs	16	8
Input definition	SMA, 50Ω termination, DC coupled	
I/P discriminator		
Threshold range	-2 V to +2 V , steps of 15 mV	
Min. pulse amplitude	50 mV typical	
Edge Detection	Rising or Falling	
Min. pulse duration (above threshold)	300 ps	
Max I/P level	-2.5 V to + 5.0V	
Counters	32-bit	
GENERAL		
Power	NIM standard from crate: + 12V @ 1.0 A & -12V @ 0.5 A OR DC power supply: 12V @ 1.5A	
Interface	USB 2.0	
OTHER I/O		
External time base input *	BNC, 10 MHz sine, 1KΩ, AC-coupled 1VPP	
Outputs		
Output definition	SMA, 50Ω termination	
Output signal	TTL pulse, >2.5 V high, 100ns duration	
Type: Pattern-detection triggered – No. of outputs	3	1
Type: Pulse generator – No. of outputs	1	

[†] Double pulse time difference to guarantee capture of all consecutive pulses. Shorter double pulse time differences will be capture probabilistically

[††] The maximum detection rate without loss of data is 25 MHz (50 MHz) with a delay of 40 µs (20 µs) due to the buffer of 1024 events / channel. Higher detection rates possible for shorter delays.

[†††] In filtering mode Logic-16 can process up to 190 Mtag/s internally to allow time of flight measurements with respect to fast laser pulses.

* Note: Use of external time base makes channel 16 (8) inactive.

3.3 Device Features

3.3.1 Active Edge and Threshold Detection

A detection event occurs when the active edge of an input signal reaches the user-defined threshold voltage. The threshold level and the active edge (rising or falling) can be adjusted for each input, individually. Logic16 uses detection events to determine when to tag time or take a count.

3.3.2 Time Tagger Mode

In this mode, Logic16 outputs a *time tag* when a detection event occurs on an input channel. This feature is useful in time correlated single photon counting especially where absolute time tags are required. This mode can be used for g2 (cross and auto correlation measurements), timing histograms, time of arrival/ time of flight, also for coincidence counting post processing. This mode could be applied in quantum communication, quantum cryptology, quantum key distribution (QKD) as well as LiDAR and other time correlation experiments.

Time is stored in the device as multiples of its internal timing resolution 156.25 ps or 78.125 ps for Configuration A or B, respectively. Note that in Logic16 there is no designated "start" channel - the reference time is that of the first tag, regardless of which input it was detected on. In other words, the first signal detected will be recorded as occurring at time = 0.

In Time Tagger Mode, Logic16 offers modifiable filtering and gating. Employing these features will improve maximum event rate, which is the rate at which assigning time tags to detection events can occur. Data transfer rate, the rate at which time tags can be sent across the USB interface, is limited to the USB connection. As a result, maximum data transfer rate is always 11 MHz.

Table 3-I summarizes the use and rates of relevant for each filtering option.

	Max. event rate	**Max. data transfer rate**	**Use when…**
	(MHz, across all inputs)		
Continuous	11	11	- All time tags must be recorded
Time Tag Filter	190	11	- Pump or trigger rate > 10 MHz, but only relevant TOF tags are needed (trigger and signal) - Measuring for long periods of time (e.g., days) and only multi-photon events are needed
Edge Gate	190	11	- Heralding events by a <= 10 MHz source
Level Gate	190	11	- Adjustable gate open with specific times

Continuous

Logic16 streams absolute time tags from ALL detected signals directly to the host PC without any filtering or gating.

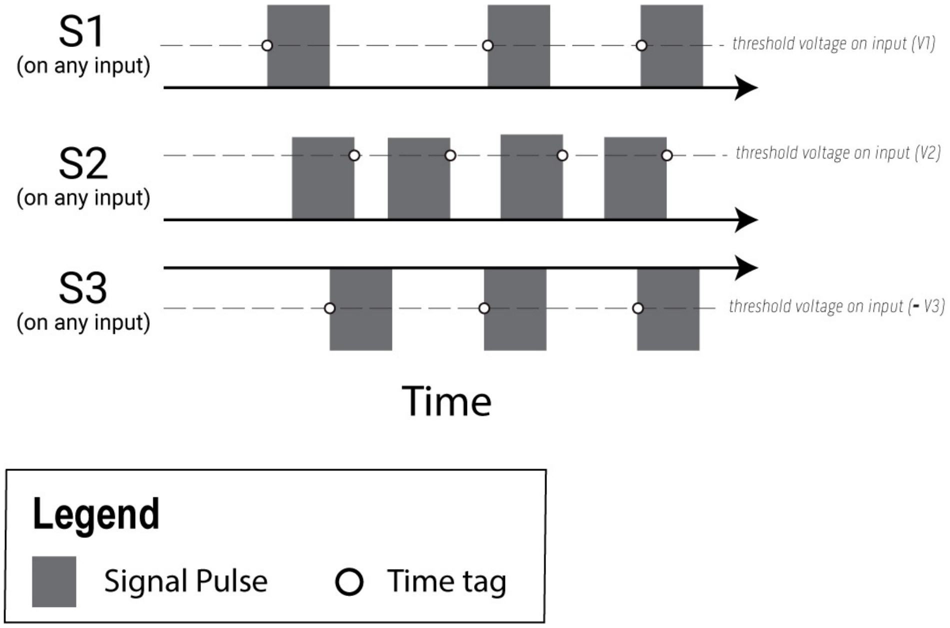

S1
(on any input)
threshold voltage on input (V1)
S2
(on any input)
threshold voltage on input (V2)
S3
(on any input)
threshold voltage on input (- V3)
Time
Legend
Signal Pulse
Time tag

*Figure 3-I **Continuous time tagging**: Three different signals, S1 (rising edge; V1), S2 (falling edge; V2) and S3 (falling edge; - V3) are time tagged concurrently. All pulses are being tagged.*

Time Tag Filter

The Time Tag Filter can be used to effectively reduce the bandwidth required to perform certain measurements. For example, it can enable time of flight (TOF) measurements with respect to fast trigger signals (up to 190 MHz), which is too high rate to be recorded in continuous time tag mode. This is possible so long as the photon detections of interest do not exceed the data transfer rate (11 MHz). The Time Tag Filter can also be used to reduce data file size when recording photon events for long periods of time (e.g., days), in cases where only multi-photon events are of interest.

The filter transmits tags only when signals appear in groups. The user must define the minimum number of tags required within a group and the maximum time that can separate adjacent time tags. A group may contain an arbitrary number of tags greater than or equal to the minimum number, as long as the time gap between any two tags is less than the defined maximum time. It is therefore important to be mindful that the maximum time must be less than the period of any periodic signal input. If the maximum time exceeds the period of a signal, all events will be tagged, as in continuous time tag mode.

Inputs can be excluded from the filter using an exception mask. In this case, the time tags on the masked channel will always be transmitted, and not included in the filter. With the time tag filter in use, Logic16 can time tag events at 190 MHz, distributed across all channels. Data transfer will still occur at up to 11 MHz.

Caution: Using filter exceptions can cause time tags not to be transmitted in correct order. (e.g. Time differences can be negative)

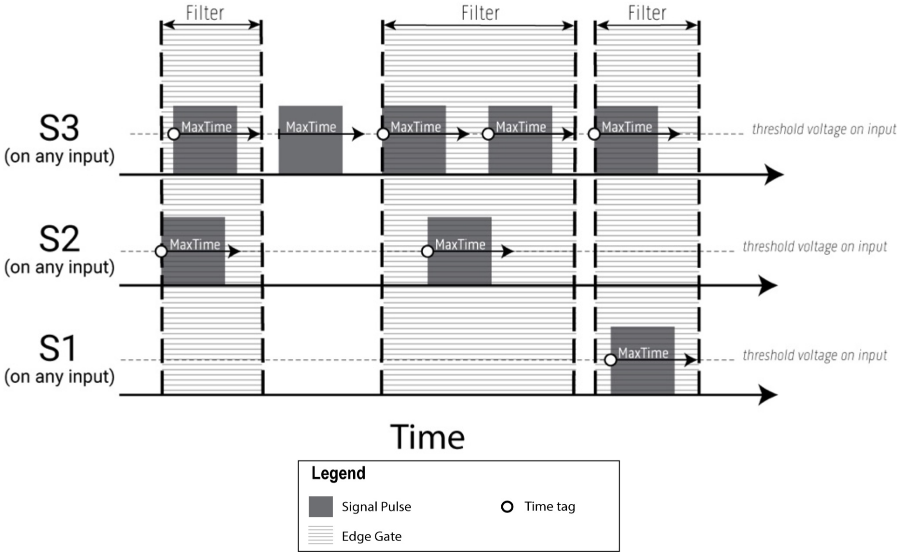


Figure 3-II ***Time Tag Filtering:*** *Signals S1, S2 and S3 are input on Logic16. With Time Tag Filter minimum group number set to 2 and maximum time as shown, only pulses with their active edge within the filter are tagged.*

Gating

Edge Gating

The Edge Gate *trigger* is located on input 8. This gate opens for a user-specified time, after there is an active edge on input 8. The gate width should be less than the period of the trigger signal or the results will be unreliable. The position of the gate can be adjusted to compensate for cable delays and similar effects. This can be a negative and positive time interval.[7]

---

[7] In configuration B, Logic16 input 8 does not time tag when using the external clock input (10 MHz). The edge gate can still be triggered on/off in this case with an input signal, however will NOT time tag.

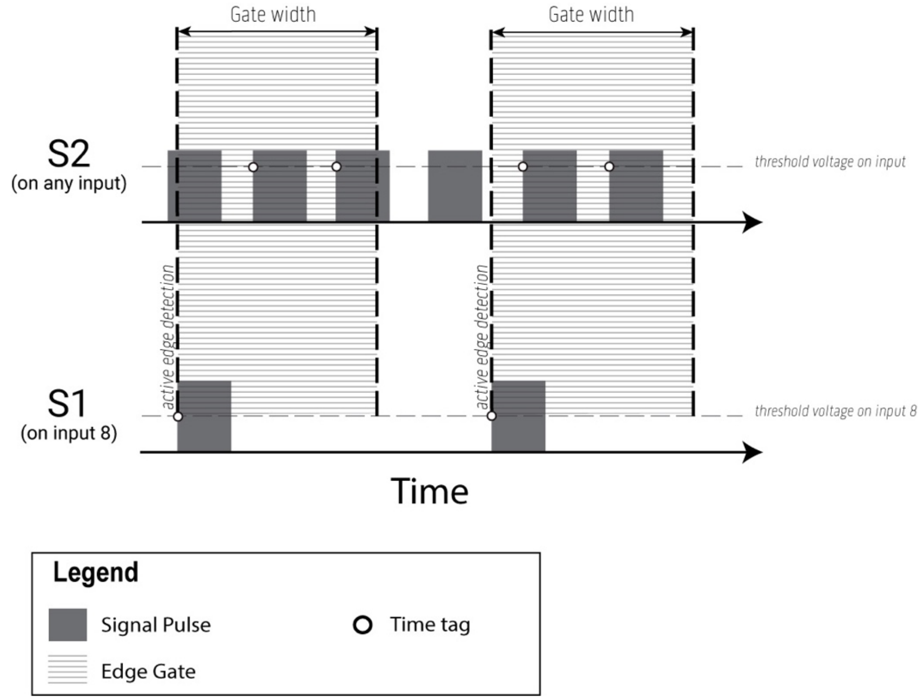


*Figure 3-III **Edge Gating**: Signal S1 on input 8 triggers the edge gate to open, when an active edge is detected. Only pulses with their active edge within the gate are being tagged*

Level Gating

The Level Gate *trigger* is located on input 9. When the signal on input 9 is higher than the specified threshold voltage, tags are processed normally. When it is lower, the input signals are ignored. Please note that the level gate signal has in internal jitter of 5 ns.

Disclaimer: Level gate starts 25ns before the trigger signal.

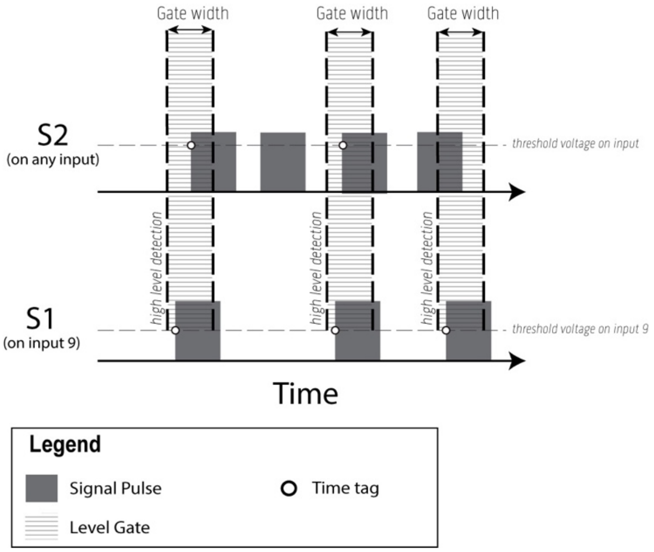


*Figure 3-IV **Level Gating:** When Signal S1 on input 9 is higher than the threshold voltage, it triggers the level gate open. Gate width is the same as the high time of S1. Only pulses with their active edge within the gate are being tagged. Note: there is an offset of ~25 ns between when the level gate starts and when trigger input is tagged.*

3.3.3 Logic mode

This mode allows users to count the occurrence of detection patterns and to output pattern-triggered TTL pulses. Coincidence counting is used for bell tests and other entanglement experiments. Pattern triggered outputs are useful with gated photon detection and telecom inGaAs detectors.

Logic Patterns (Coincidence Counting)

Logic Pattern Detection events are counted when input pattern conditions are met within the coincidence window.

The **Patterns** are defined by choosing each input condition to be one of:

- **+** **active:** true if the input has an active edge within the coincidence window
- **−** **inactive:** true if the input does NOT have an active edge within the coincidence window while there is at least one active edge on another input
- **Ø** **ignore:** true regardless of whether the input has an active edge or not

Each pattern must have a $(+)$ on at least one input, otherwise it is ill-defined. Multiple patterns can be counted concurrently on-board the unit.

The **coincidence window** is the *maximum* time from the first $(+)$ detection to the last[8] $(+)$ in the pattern detection event. When all logically selected conditions $(+, -, \text{Ø})$ are met within that time window, the event is counted. Otherwise the event is not counted.

The user can modify the coincidence window (see specifications). Ideally there will not be more than one (+) on a single input, within one coincidence window. If this occurs, a “DoubleError” flag will be thrown.

Examples

In the descriptions below, M is the number of effective input channels 16 (8) for configuration A (B).

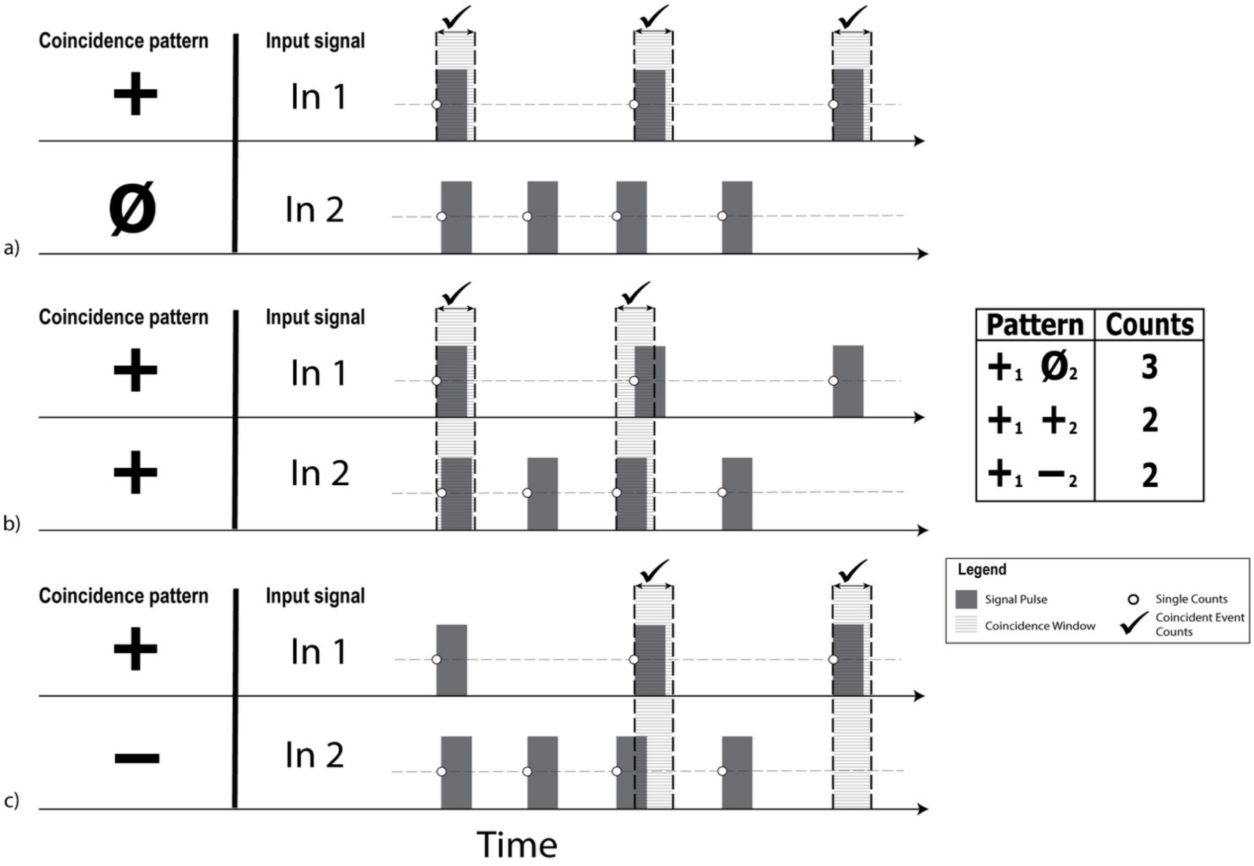
An example **singles** event $(+_1, \text{Ø}_2 \ldots \text{Ø}_M)$ is shown in Figure 3-V.a: the pattern event is counted every time there is an active edge $(+)$ on input 1, while all other inputs are ignored.

An example **coincidence** event $(+_1, +_2, \text{Ø}_3 \ldots \text{Ø}_M)$ is shown in Figure 3-V.b: the pattern event is counted when both inputs 1 and 2 have an active edge within the same coincidence window.

An example **anticoincidence** event $(+_1, -_2, \text{Ø}_3 \ldots \text{Ø}_M)$ is shown in Figure 3-V.c: the pattern event is counted when there is an active edge on input 1 but not input 2, within the coincidence window.

---

[8] If it is a singles event, the first and last detection are one and the same



Pattern	Counts
$+_1$ Ø$_2$	3
$+_1$ $+_2$	2
$+_1$ $-_2$	2

Figure 3-V ***Pattern Processed Coincidence Counting:*** *For the same two signal (on input 1 and 2), three pattern events are counted. Here it is assumed that channels 3 to M are all set to Ø.*

Pattern-Triggered Outputs

In Logic mode, the user can generate up to 3 (1) pattern-triggered outputs in Configuration A (B)[9]. The output channels are accessible on the front panel of the unit (Output 1 – 3). These outputs are triggered by detection events as defined in the Logic Pattern section, however, instead of counting these patterns when they occur, registration of an event will trigger the output of a TTL pulse. Each output can be set to its own, distinct pattern trigger.

Note that these patterns are also set *independent* from coincidence counting patterns. In other words, the user can set any pattern to be counted and any other pattern to trigger an output. Counting and pattern-triggered outputs can be used simultaneously.

For each event, Logic16 outputs 2.5 V high TTL pulses with a default delay of 350 ns from the first detection within the coincidence window. The pulse width is adjustable.

---

[9] In configuration B, Outputs 2 and 3 are not operational

3.3.4 Delay

Logic16 enables user to define individual delays for each input channel. This feature is very useful in the logic mode, where the delays of individual channels can be fine-tuned to maximize coincidence signals.

3.3.5 Function Generator

Logic16 has its own simple function generator located on Output 4. Frequency and duty cycle can be adjusted. The cycle time and high time can be adjusted in 5 ns increments, from 10 ns to 1.3 sec.

Note: The function generator always uses the internal frequency reference and not the external 10 MHz signal.

3.3.6 External Clock Input

The external clock 10MHz input can be used for local synchronization to access more change for even larger multi-photon experiments as well as long distance synchronization, e.g. in quantum communication (including Quantum cryptography and quantum key distribution).

The 10 MHz input is located on the rear side of the device. It can be used to increase the stability of long-term measurements by connecting a stable clock source. The input is AC coupled. For this reason it can be driven by a variety of digital signals. As long as the corresponding 10 MHz error flags do not raise, the time base can be considered as valid. When using digital signals the rise and fall time should be greater than 5 ns to avoid ringing.

Levels: There are two possibilities to drive the 10 MHz input:

	Level	**Termination**
Sinus	1 Vpp nominal	1 kΩ internal
Digital	Minimum level 500 mVpp	External 50 Ω termination required

Note: Use of external time base makes channel 16 inactive in configuration A and channel 8 inactive in configuration B.

3.4 Firmware Update/Change

The following section outlines how to perform firmware updates and changes. For best results it is important that the Logic16 Unit is running the latest FPGA firmware. The same procedure for updates is used to switch from Configuration A to B, or vice versa. *Tip: You can check the firmware version quickly using Timetag Explorer*[10].

You need the following items:

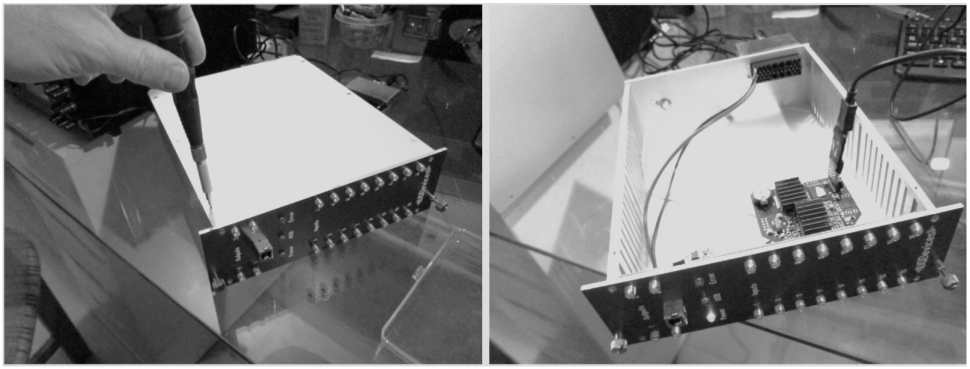
- ✓ Logic16 Unit
- ✓ Small Phillips or flathead screw driver
- ✓ JTAG link cable (e.g. JTAG-cable Xilinx, Xilinx parallel cable, HS2 Digilent)
- ✓ Xilinx Lab Tools – iMPACT (Free download from http://www.xilinx.com/downloads)
  - o Find and download latest ISE Design Tools available for your PC under Vivado (HW Developer) in ISE Archive.
  - o Follow setup instructions given by Xilinx and ensure Xilinx iMPACT is accessible.
- ✓ Firmware PROM files provided by UQD (found in the CD drive provided)
- ✓ Time Tag Explorer (*optional)*

Perform the following procedures to update/change the Firmware:

i Connect Logic16 to PC with a JTAG Cable

Modules manufactured *after* January 2015: connect JTAG Cable to the designated JTAG connector found on the rear of the device.

Modules manufactured *before* January 2015: remove the right-hand side panel (unscrew). Connect the JTAG Cable to the FPGA board and then to your computer.



ii Connect device to a power supply.

Connect Logic16 to a DC power supply (12 V @ ≥1.5 A) or a NIM crate (12 V @ 1.0 A and -12V @0.4 A). The ports are found on the rear side of the device.

---

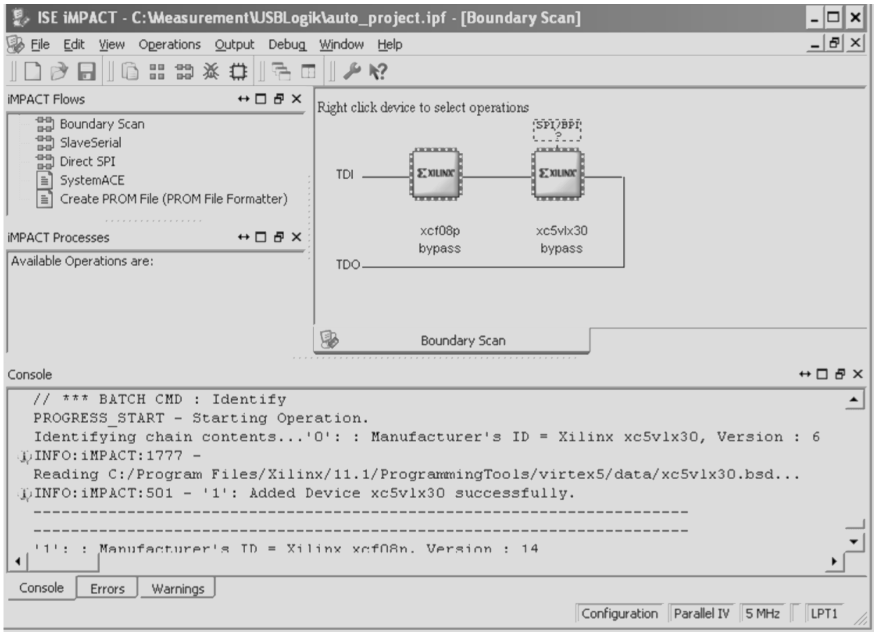
[10] Upon connection, it will appear in the window on the Time Tag Explorer's TimeTag tab

iii Run the program “Xilinx iMPACT” and connect iMPACT to device

Once the program is running, a prompt will appear to create a new project. Click “yes”. Select “Configure devices using Boundary-Scan (JTAG)” and then click “OK”. Once connection has been established with Logic16, another prompt will appear to Auto Assign the configuration files. Click “No”. Choose “Cancel” if a further window appears.
Note: An error may pop up if Logic16 is not powered on. Make sure power LED is on when trying to connect to the device.

iv Assign new configuration file

iMPACT will display two devices. Only the first will have its firmware updated. Right click on the first device (xcf08p) and choose “Assign new configuration file”. Choose the desired file .mcs file under “Firmware” from the CD drive provided with the device. i.e. “Timetag_230_06_highres.mcs”. Once selected, right click again on this device and choose “Program”. The device programming properties will open. Select the checkmark beside "Verify" and click "OK". Once done, iMPACT will display “Program Succeeded”.


ISE iMPACT - C:\Measurement\USBLogik\auto_project.ipf - [Boundary Scan]
File Edit View Operations Output Debug Window Help
iMPACT Flows
Boundary Scan
SlaveSerial
Direct SPI
SystemACE
Create PROM File (PROM File Formatter)
iMPACT Processes
Available Operations are:
Right click device to select operations
SPI/BPI
?
TDI
TDO
xcf08p
bypass
xc5vlx30
bypass
Boundary Scan
Console
Console
Errors
Warnings
Configuration
Parallel IV
5 MHz
LPT1

```
// *** BATCH CMD : Identify
PROGRESS_START - Starting Operation.
Identifying chain contents...'0': : Manufacturer's ID = Xilinx xc5vlx30, Version : 6
INFO:iMPACT:1777 -
Reading C:/Program Files/Xilinx/11.1/ProgrammingTools/virtex5/data/xc5vlx30.bsd...
INFO:iMPACT:501 - '1': Added Device xc5vlx30 successfully.
----------------------------------------------------------------------
----------------------------------------------------------------------
'1': : Manufacturer's ID = Xilinx xcf08p, Version : 14
```

v Close iMPACT and disconnect Logic16

Close iMPACT, power down the device, and disconnect the Xilinx Parallel Cable. If the side panel had been detached for this process, secure it back on the device. The firmware update is complete.

vi Reboot and confirm firmware update was correct

Reconnect power to the Logic Unit to reboot the device, then check firmware version with Time Tag Explorer *(or your own program).*

Section 4

Software Interface

4.1 Drivers Installation

Logic16 comes with .NET drivers for Windows and C++ drivers for Windows and Linux.

4.1.1 Windows 8 or higher

Devices with FPGA version greater than 2.17 do not need a driver installed on Windows 8.

Plug in the device and the driver loads automatically.

Caution: Do not connect a device with firmware version below 2.17 to a computer running Windows 8. Windows stores the presence of an automatic driver in the registry. When it doesn't find a driver on the first attempt, it will never ask the device again.

4.1.2 Pre Windows 8

i. Extract the Zip file with the USB driver
ii. Plug the unit into the NIM crate or DC power supply
iii. Connect the unit to the computer using a USB cable
iv. Power on the NIM crate
v. When prompted, choose manual driver selection
vi. Navigate to the folder with the extracted files and open the sub folder "USB Driver"
vii. The driver is installed and the device is ready for use.

4.1.3 Linux

Linux C++ drivers for Intel processor 32 and 64 bit are provided. Linux 32 bit on ARM is provided for test purposes and is not guaranteed to run on all ARM environments.

On Linux, copy libtimetag32.so or libtimetag64.so to your lib directory. If you have not already done so, get the g++ compiler and libusb development version.

```
sudo apt-get install g++-4.6-multilib

sudo apt-get install libusb-1.0-0-dev
```

4.2 Graphical User Interface

UQDevices provides two different GUI for users for ease of using Logic16: Timetag Explorer or UQDevices Logic16 Correlation Counter. They both use the ttinterface.dll file provided in the CD driver to operate.

4.2.1 Timetag Explorer

TimeTag explorer provides a general interface where you can access all functionalities of the device as well as collect basic data without the need for coding.

Setup of Timetag Explorer

Prerequisites

Before installing Timetag explorer, please take sure that the following components are installed on your computer:

- Microsoft .NET Framework 4.0 (The "Client Profile" version will do)
  It can be downloaded here:
  `www.microsoft.com/en-US/download/details.aspx?id=17718`
- Microsoft Visual C++ 2010 Redistributable Package
  It can be downloaded here:
  `www.microsoft.com/en-US/download/details.aspx?id=5555`

NOTE: The links in the start menu are renamed each installation. You can have several versions of TimeTag Explorer and ttInterface.dll at the same time.

Installation

We recommend using the latest version of Time Tag Explorer found within CD driver provided. It is located under Applications > TimeTagExplorer. Choose the Release folder that fits your PC (34 bit or 64 bit). Open up Time Tag Explorer by click on the .exe file.

Older versions of the software are located within "Legacy" folder under TimeTagExplorer.

Using Timetag Explorer

Connect

Once the Time Explorer window is open, click connect to establish a connection with the device. The USB LED on the device should light up. Note: all other functions is disabled until a connection is achieved.

Calibrate

Click "Calibrate" to calibrate the unit. This operation needs about 4-10 seconds to complete. Calibration should be done before any other feature is selected.

Note: Intrinsic delays of each input may vary after the calibration procedure is performed.

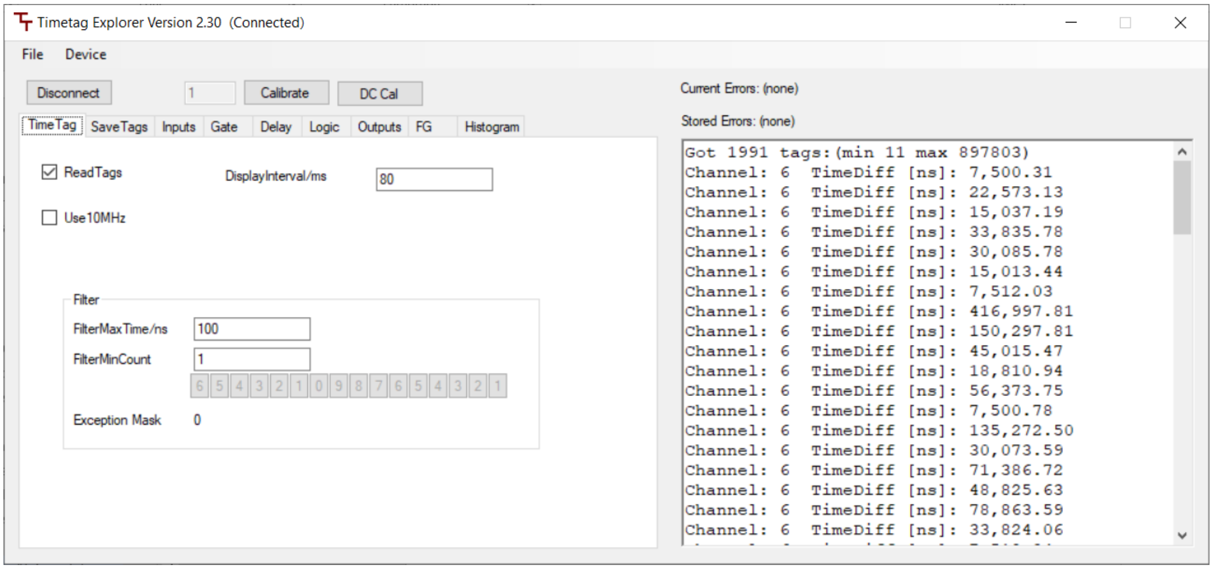
DC Calibrate

DC calibration is a useful feature when triggering very small signals near zero. DC Calibration only has to be done only once for each unit. Before starting DC calibration please make sure that all inputs have zero voltage. This is done best by disconnecting all SMA connectors.

- Then press the button "DC Cal".
- The following dialog appears. Press "Ok"

After a few seconds the calibration is completed and the calibration data is saved to disk.

The unit number is encoded in the file name. When there are several units connected to one computer, each unit load its correct correlation data. This is true as long as you don't change the USB wiring.



1

☑ ReadTags

DisplayInterval/ms 80

☐ Use10MHz

FilterMaxTime/ns 100

FilterMinCount 1

Exception Mask 0

Current Errors: (none)

Stored Errors: (none)

```
Got 1991 tags:(min 11 max 897803)
Channel: 6  TimeDiff [ns]: 7,500.31
Channel: 6  TimeDiff [ns]: 22,573.13
Channel: 6  TimeDiff [ns]: 15,037.19
Channel: 6  TimeDiff [ns]: 33,835.78
Channel: 6  TimeDiff [ns]: 30,085.78
Channel: 6  TimeDiff [ns]: 15,013.44
Channel: 6  TimeDiff [ns]: 7,512.03
Channel: 6  TimeDiff [ns]: 416,997.81
Channel: 6  TimeDiff [ns]: 150,297.81
Channel: 6  TimeDiff [ns]: 45,015.47
Channel: 6  TimeDiff [ns]: 18,810.94
Channel: 6  TimeDiff [ns]: 56,373.75
Channel: 6  TimeDiff [ns]: 7,500.78
Channel: 6  TimeDiff [ns]: 135,272.50
Channel: 6  TimeDiff [ns]: 30,073.59
Channel: 6  TimeDiff [ns]: 71,386.72
Channel: 6  TimeDiff [ns]: 48,825.63
Channel: 6  TimeDiff [ns]: 78,863.59
Channel: 6  TimeDiff [ns]: 33,824.06
```

Figure 4-I Time Tag Tab in Time Tag Explorer

TimeTag tab

This tab contains the very basic operation modes.

✓ `ReadTags`

Select “ReadTags” to start reading tags. The right hand side window displays the channel number and the time difference to the previous time value. The first tagged received is time zero.

☐ `Use10MHz`

Select this to connect to the external timing reference. When no appropriate signal is connected on the rear side of the device, an error flag is shown.

☐ `Display Interval/ms`
Enter the value of time interval (in ms) you would like to observe on the display window.

☐ `Filter`

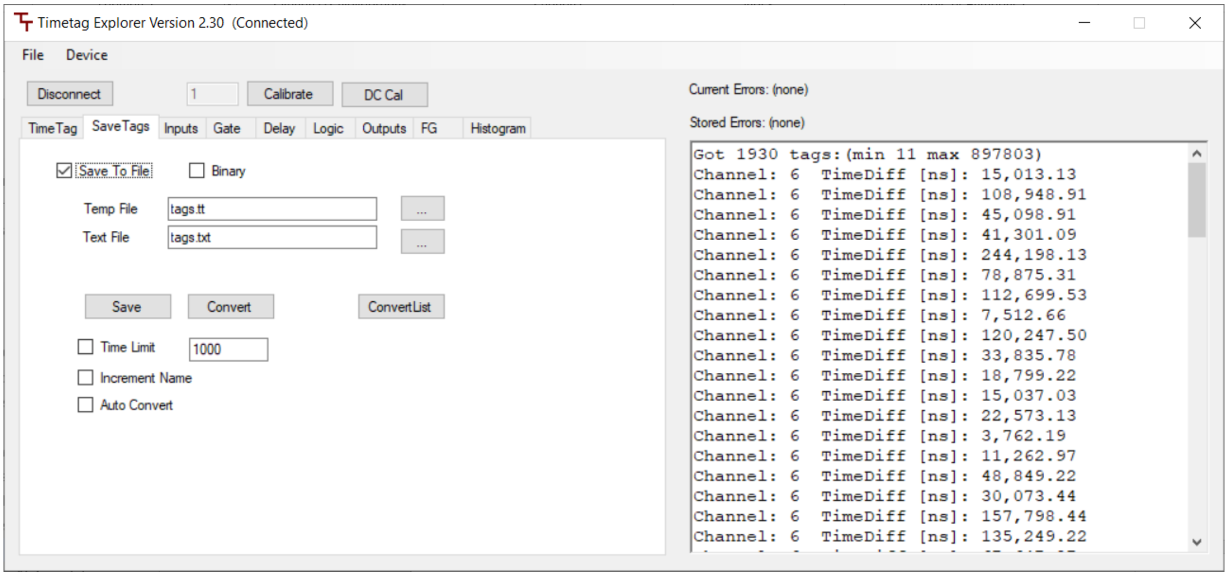
`FilterMinCount:`	Enter the minimum number of tags required in one group. When the value is 1, all tags are transmitted.
`FilterMaxTime:`	Enter the maximum time interval (ns) between any two tags within a group. Ensure the value is less than period of any of the input signals, or else all tags will be transmitted.

Note: A group be of arbitrary length as long as there is no time gap greater than the specified maximum time.

`FilterExceptions:` Click on the corresponding channel number to mark them with “+”. All Inputs marked with a “+”are excluded from the filter. They are always transmitted. (e.g. 1pps pulse)

Caution: Using filter exceptions can cause time tags not to be transmitted in correct order. (e.g. Time differences can be negative)

`ExceptionMask:` This value is changes automatically when Filter Exceptions is changed. This is the value to be transmitted to SetFilterExceptions().



Current Errors: (none)
Stored Errors: (none)

1

☑ Save To File ☐ Binary

Temp File: tags.tt
Text File: tags.txt

☐ Time Limit 1000
☐ Increment Name
☐ Auto Convert

```
Got 1930 tags:(min 11 max 897803)
Channel: 6  TimeDiff [ns]: 15,013.13
Channel: 6  TimeDiff [ns]: 108,948.91
Channel: 6  TimeDiff [ns]: 45,098.91
Channel: 6  TimeDiff [ns]: 41,301.09
Channel: 6  TimeDiff [ns]: 244,198.13
Channel: 6  TimeDiff [ns]: 78,875.31
Channel: 6  TimeDiff [ns]: 112,699.53
Channel: 6  TimeDiff [ns]: 7,512.66
Channel: 6  TimeDiff [ns]: 120,247.50
Channel: 6  TimeDiff [ns]: 33,835.78
Channel: 6  TimeDiff [ns]: 18,799.22
Channel: 6  TimeDiff [ns]: 15,037.03
Channel: 6  TimeDiff [ns]: 22,573.13
Channel: 6  TimeDiff [ns]: 3,762.19
Channel: 6  TimeDiff [ns]: 11,262.97
Channel: 6  TimeDiff [ns]: 48,849.22
Channel: 6  TimeDiff [ns]: 30,073.44
Channel: 6  TimeDiff [ns]: 157,798.44
Channel: 6  TimeDiff [ns]: 135,249.22
```

Figure 4-II Save Tag tab

SaveTags tab

This tab can be used to store the time tags to an ACSII file. To improve the transmission rate, the data is written to a temporary file first. This way the full data rate can be saved to disk. The names of the temporary file and the output file can be entered in the two corresponding text boxes. The path can be a relative or an absolute path. When the directory is not present, it is created.

Clicking the “Save” button initiates the capturing of the data. Data is saved until the “Stop” button is pressed. Alternatively one can enter a time limit. In that case data is saved for a predefined amount of time, e.g. 2 sec. After pressing “Convert” the temporary file is converted to a text file.

By choosing “IncrementName” the temporary filename is automatically changed each time “Stop” is pressed. This features is particularly useful in conjunction with the “ConvertList” feature.

The “Convert List” feature automatically converts all temporary files in a certain folder. The folder and the file extension are taken from the “Temp File” text box. The text box just has to link to one temporary file. “Convert List” automatically converts all files.This feature is also quite helpful when saving a list of temporary files using the API (e.g. using LabVIEW)

File Format

The data can be stored in text or binary data format.

Text Format: The converted text file contains the channel number [1...16] and the corresponding time. They are separated by a tab character. The time is stored as multiples of the internal timing resolution. The reference time is the time of the first tag. In case of an overflow, a special tag 28 is inserted in the data stream.

Example: (1 MHz on channel 1)

```
1	0
1	6399
1	12799
1	19199
1	25598
1	31998
1	38397
1	44797
1	51197
1	57595
1	63995
1	70394
1	76794
```

Binary file format: The binary format contains a series of 9 byte records. The first byte is the channel in the range of [1...16]. The next 8 bytes contain a long integer containing the time in internal units. The time starts at 0.

Raw file format: The user can choose to process the raw file format without converting it. Note, this format could change in future versions. The raw file format is a series of 32 bit integers. Each integer is divided in two fields:

Channel: The cannel is encoded in the bits [31...26]. It is in the range of [0...15]. Values higher than 15 have a special meaning, see below.

Special channel numbers

    0x1f Dummy - Please ignore
    0x1e High bit - Bits [51...26] of the following tag
        0x1d Overflow - At this position some data is missing
        0x10-0x1c Reserved - Please ignore

Time: The low bit of the time value are encoded in the bits [25...0]. The high bits of the time value have to be reconstructed by user software.

Input Tab

☐ `Name`

In this text box a name can be assigned to each input. This is optional.

☐ `Level`

This is the threshold voltage of the input.

☐ `Edge`

The “Edge” checkbox toggles the relevant edge. If not selected, the positive edge is relevant.

☐ `Active`

When the input has an active level, the color of this field turns to blue. The active level depends on the selected edge. When using positive edge, the active level is high.

☐ `Input Frequency`

This shows the single frequency of the respective input. This value is displayed regardless whether the “logic” functionality is enabled or not. Please note: The two sets of values are measured over a different time period and therefore the result can differ slightly.

Logic Tab

This tab is only available with the UQD devices that include the special “Logic” feature. It allows testing and studying of some of the coincidence features of the logic, like measuring single and coincident detections from arbitrary channels, and adjusting delays between channels. The output on the right side gives the single count rates per cycle time and in thousands per second (`kHz`) for each `Input` channel and, below, the count rates for the arbitrary coincidence `patterns`.

☐ `Use Logic`

Check off to start using logic mode

☐ `Store MinMax`

Check off to show maximum/minimum kHz

☐ `Approx. Cycle time`

The software tries to read the data with this measurement time. Because of computer processing time, the actual cycle time is different. This time Cycle/ms is displayed on the right and changes slightly. Only the actual cycle time is used to calculate detection rates.

☐ `Coincidence Window`

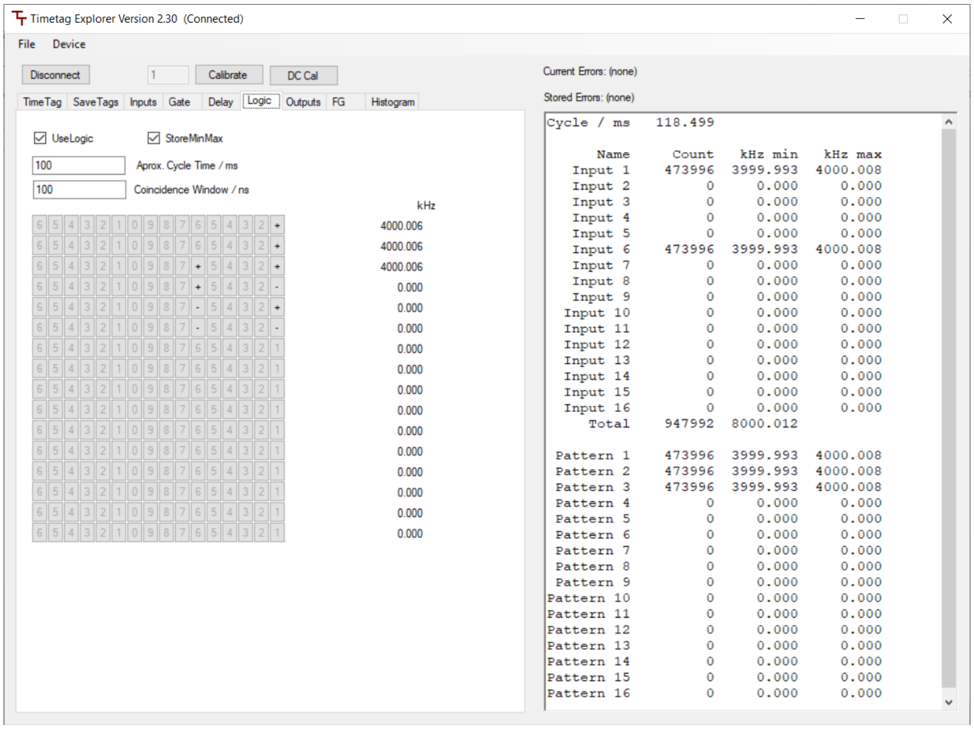
Enter the maximum time from the first detection to the last detection in the coincidence event. When all logically selected detections occur within that time window, then the event is counted. Otherwise the event is not counted.

☐ `Coincidence Patterns`

Each line of buttons defines one coincidence event. Clicking on the button changes the state of the corresponding input. You can set up to 16 patterns simultaneously.

+ The input must have an active edge in the coincidence window for the event to be counted.

- The input must not have an active edge in the coincidence window for the event to be counted.



Current Errors: (none)

Stored Errors: (none)

☑ UseLogic ☑ StoreMinMax

100 Aprox. Cycle Time / ms

100 Coincidence Window / ns

```
                                               kHz
6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 +            4000.006
6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 +            4000.006
6 5 4 3 2 1 0 9 8 7 + 5 4 3 2 +            4000.006
6 5 4 3 2 1 0 9 8 7 + 5 4 3 2 -               0.000
6 5 4 3 2 1 0 9 8 7 - 5 4 3 2 +               0.000
6 5 4 3 2 1 0 9 8 7 - 5 4 3 2 -               0.000
6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1               0.000
6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1               0.000
6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1               0.000
6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1               0.000
6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1               0.000
6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1               0.000
6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1               0.000
6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1               0.000
6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1               0.000
6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1               0.000
```

```
Cycle / ms   118.499

      Name      Count   kHz min   kHz max
   Input 1     473996  3999.993  4000.008
   Input 2          0     0.000     0.000
   Input 3          0     0.000     0.000
   Input 4          0     0.000     0.000
   Input 5          0     0.000     0.000
   Input 6     473996  3999.993  4000.008
   Input 7          0     0.000     0.000
   Input 8          0     0.000     0.000
   Input 9          0     0.000     0.000
  Input 10          0     0.000     0.000
  Input 11          0     0.000     0.000
  Input 12          0     0.000     0.000
  Input 13          0     0.000     0.000
  Input 14          0     0.000     0.000
  Input 15          0     0.000     0.000
  Input 16          0     0.000     0.000
     Total     947992  8000.012

 Pattern 1     473996  3999.993  4000.008
 Pattern 2     473996  3999.993  4000.008
 Pattern 3     473996  3999.993  4000.008
 Pattern 4          0     0.000     0.000
 Pattern 5          0     0.000     0.000
 Pattern 6          0     0.000     0.000
 Pattern 7          0     0.000     0.000
 Pattern 8          0     0.000     0.000
 Pattern 9          0     0.000     0.000
Pattern 10          0     0.000     0.000
Pattern 11          0     0.000     0.000
Pattern 12          0     0.000     0.000
Pattern 13          0     0.000     0.000
Pattern 14          0     0.000     0.000
Pattern 15          0     0.000     0.000
Pattern 16          0     0.000     0.000
```

Figure 4-III Time Tag Explorer Logic Tab: showing single and coincidence counts

Output Tab

This tab lets user access the Outputs 1-3 (configuration A) or Output 1 (configuration B) on the device. UseLogic must be checked off in logic tab first in order to access Output functions.

☐ `Output Width/ns`

Enter the value of desired pulse high time in ns.

☐ `Output Event Count`

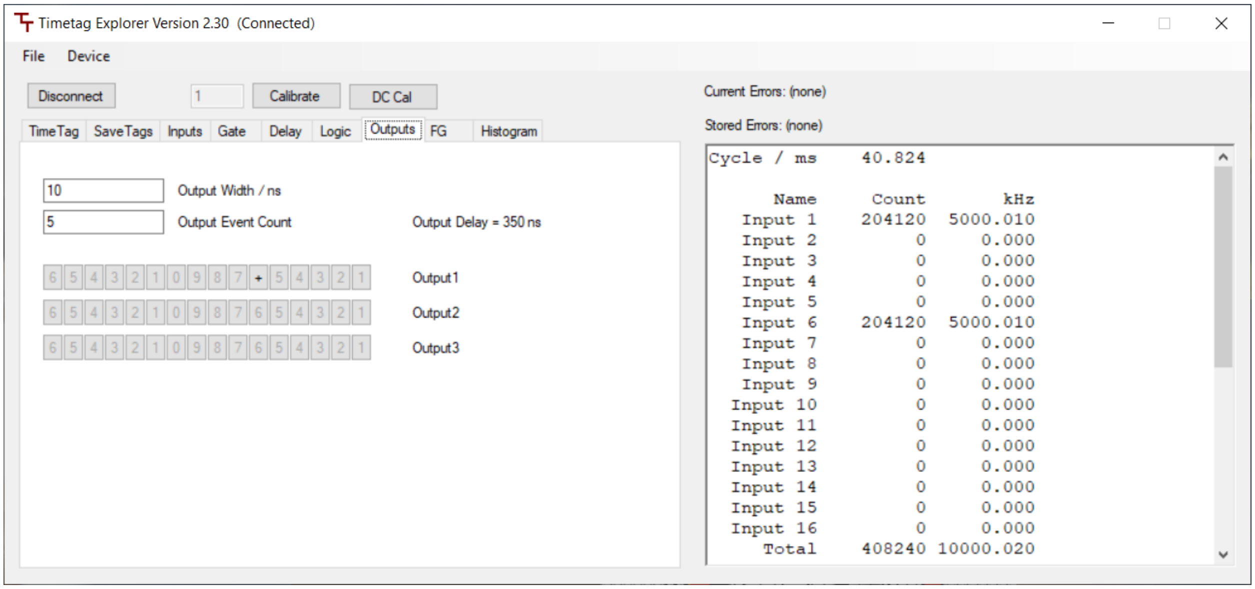
Enter the number of events you expect to occur in one 5 ns time slice under worst-case conditions. Increasing the value by one will increase the delay of the outputs by 10 ns.

☐ `Output Coincidence Pattern`

Set coincidence pattern by clicking on the button to change the state of the corresponding input.

+ The input must have an active edge in the coincidence window for the event to be counted.

- The input must not have an active edge in the coincidence window for the event to be counted.



Output Width / ns: 10

Output Event Count: 5

Output Delay = 350 ns

Current Errors: (none)

Stored Errors: (none)

```
Cycle / ms    40.824

     Name      Count        kHz
  Input 1     204120   5000.010
  Input 2          0      0.000
  Input 3          0      0.000
  Input 4          0      0.000
  Input 5          0      0.000
  Input 6     204120   5000.010
  Input 7          0      0.000
  Input 8          0      0.000
  Input 9          0      0.000
 Input 10          0      0.000
 Input 11          0      0.000
 Input 12          0      0.000
 Input 13          0      0.000
 Input 14          0      0.000
 Input 15          0      0.000
 Input 16          0      0.000
    Total     408240  10000.020
```

Figure 4-IV Output tab displaying pattern-triggered outputs

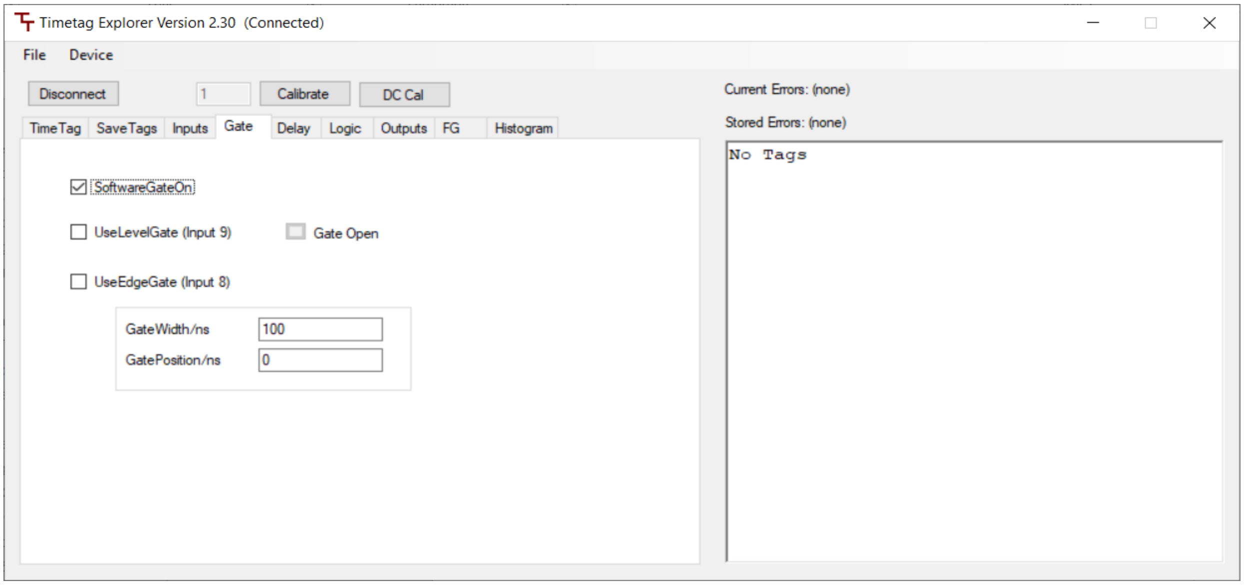
Gate Tab

☐ `SoftwareGateOn`
Software gate should be activated at all times.

☐ `UseLevelGate`
Check off to start level gating. The checkbox `LevelGateActive` is automatically checked, when the gate signal is high.

☐ `UseEdgeGate`
Check off to start edge gating. Specify the preferred length of in `GateWidth` (ns). The position of the gate can be adjusted to compensate cable delays and similar effects.

Note: that the gate position can be a negative and positive time interval.



☑ SoftwareGateOn

☐ UseLevelGate (Input 9) ☐ Gate Open

☐ UseEdgeGate (Input 8)

GateWidth/ns: 100

GatePosition/ns: 0

Current Errors: (none)

Stored Errors: (none)

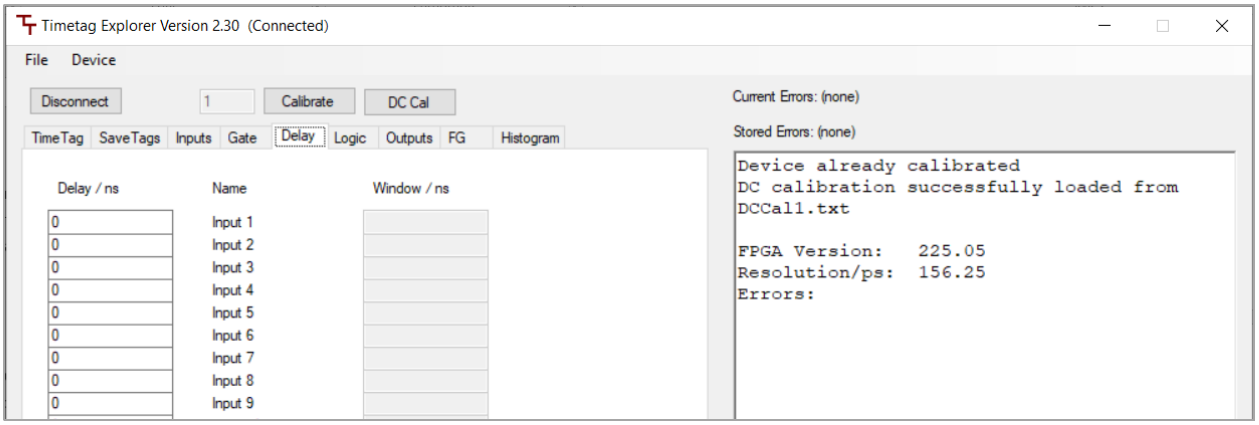
```
No Tags
```

Figure 4-V Gate Tab

Delay Tab

This tab allows the user to define individual delays for each input channel.

This feature is very useful in the logic mode, where the delays of individual channels can be fine-tuned to maximize coincidence signals.



Delay / ns	Name	Window / ns
0	Input 1	
0	Input 2	
0	Input 3	
0	Input 4	
0	Input 5	
0	Input 6	
0	Input 7	
0	Input 8	
0	Input 9	

Current Errors: (none)

Stored Errors: (none)

```
Device already calibrated
DC calibration successfully loaded from
DCCal1.txt

FPGA Version:   225.05
Resolution/ps:  156.25
Errors:
```

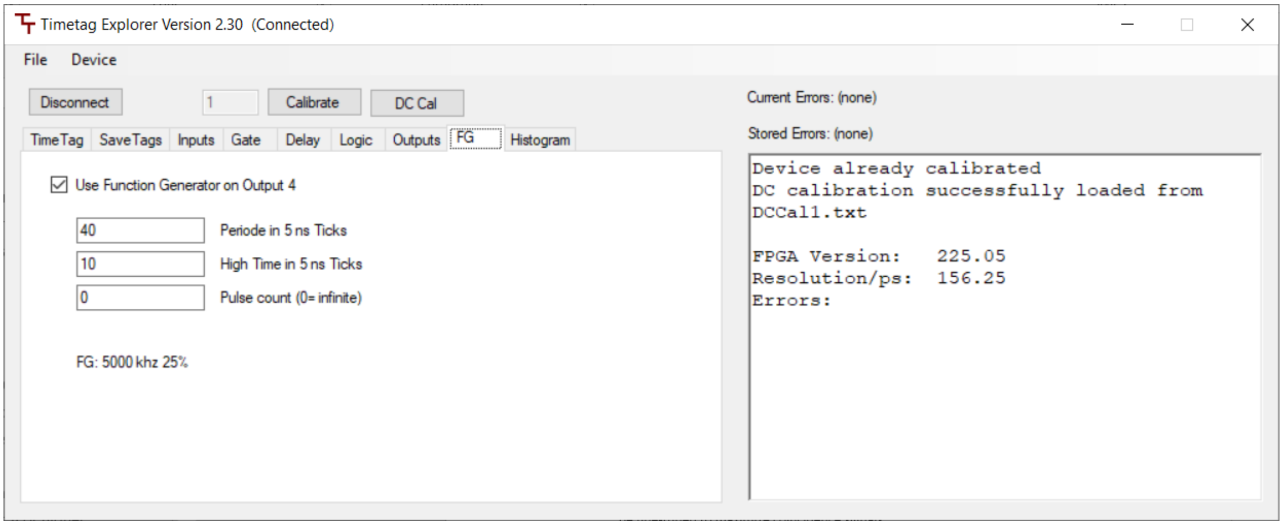
Figure 4-VI Delay Tab

FG Tab

This tab allows user to access the function generator located on Output 4. The cycle time and high time can be adjusted in 5 ns increments. The actual frequency and duty cycle is calculated and output below the text boxes. The values can be adjusted from 10 ns to 1.3 sec.

- ☐ `Period`
  Enter period of pulses in 5 ns increments
- ☐ `High time`
  Enter pulse width required in 5 ns increments
- ☐ `Pulse count`
  Enter the number of pulses required. 0 generates infinite pulses.

Note: The function generator always uses the internal frequency reference and not the external 10 MHz signal.



☑ Use Function Generator on Output 4

40 Periode in 5 ns Ticks

10 High Time in 5 ns Ticks

0 Pulse count (0= infinite)

FG: 5000 khz 25%

Current Errors: (none)

Stored Errors: (none)

```
Device already calibrated
DC calibration successfully loaded from
DCCal1.txt

FPGA Version:   225.05
Resolution/ps:  156.25
Errors:
```

Figure 4-VII Function Generator

Histogram

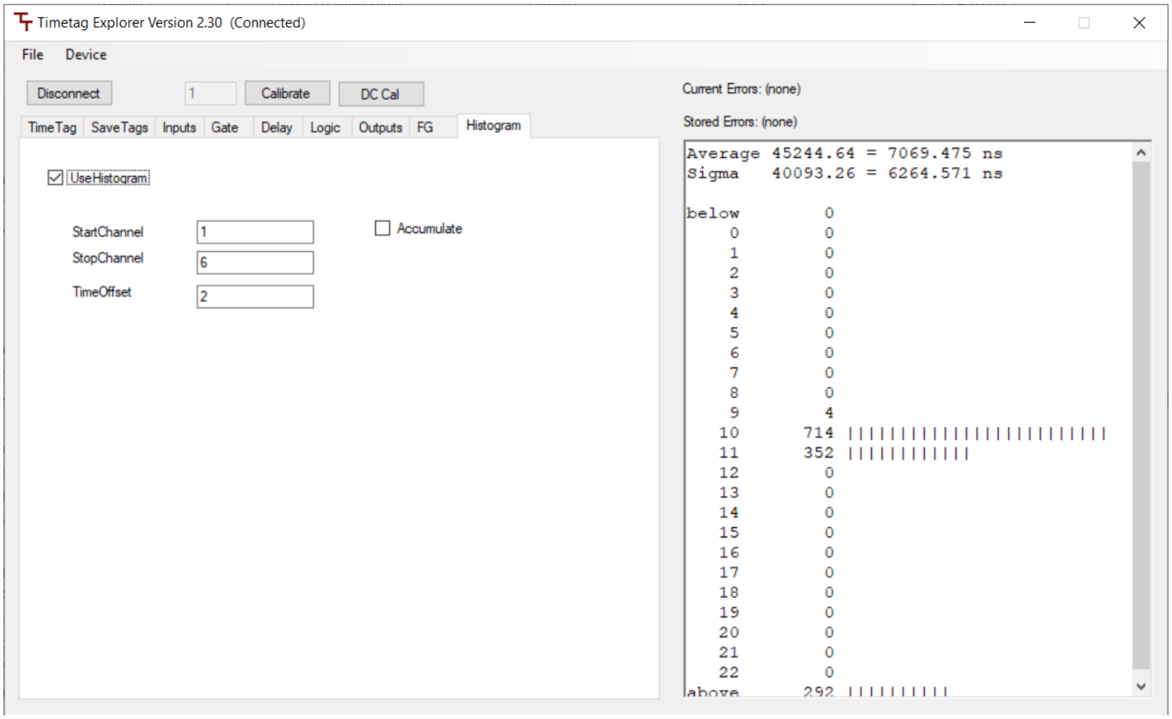
- ☐ `Use Histogram`
  Check off to start creating a histogram
- ☐ `Accumulate`

  Check off for a cumulative histogram
- ☐ `Start Channel`

  Enter the start channel #
- ☐ `Stop Channel`

  Enter the stop channel #
- ☐ `Time offset`

  Enter amount of time to offset in resolution time increments.



☑ UseHistogram

StartChannel 1

StopChannel 6

TimeOffset 2

☐ Accumulate

Current Errors: (none)

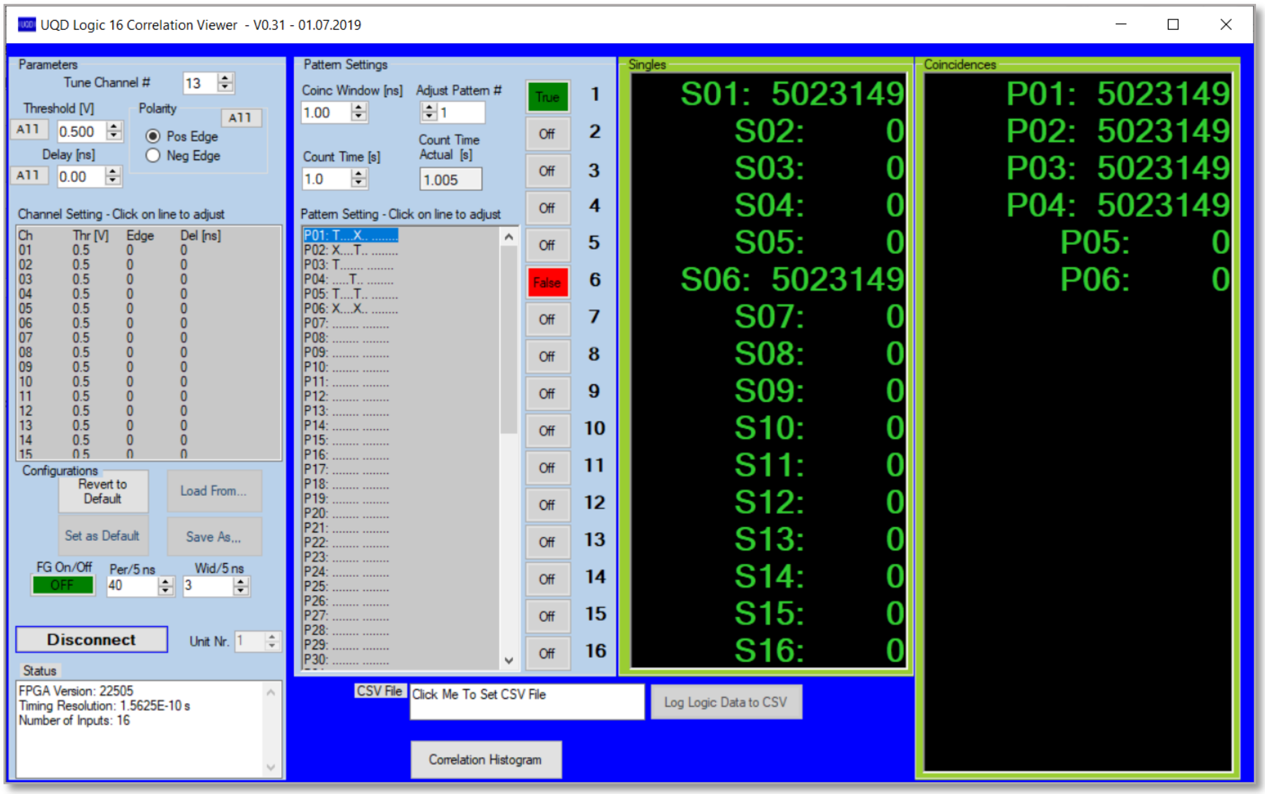
Stored Errors: (none)

```
Average 45244.64 = 7069.475 ns
Sigma   40093.26 = 6264.571 ns

below        0
    0        0
    1        0
    2        0
    3        0
    4        0
    5        0
    6        0
    7        0
    8        0
    9        4
   10      714 |||||||||||||||||||||||||
   11      352 ||||||||||||
   12        0
   13        0
   14        0
   15        0
   16        0
   17        0
   18        0
   19        0
   20        0
   21        0
   22        0
above      292 ||||||||||
```

Figure 4-VIII Histogram Tab

4.2.2 UQD Logic16 Correlation Viewer

Correlation Viewer provides a general interface where you can access Logic Mode directly. This is helpful if you require coincidence counting.



UQD Logic 16 Correlation Viewer - V0.31 - 01.07.2019

Parameters

Tune Channel # 13

Threshold [V] 0.500

Delay [ns] 0.00

Polarity: ● Pos Edge ○ Neg Edge

Channel Setting - Click on line to adjust

Ch	Thr [V]	Edge	Del [ns]
01	0.5	0	0
02	0.5	0	0
03	0.5	0	0
04	0.5	0	0
05	0.5	0	0
06	0.5	0	0
07	0.5	0	0
08	0.5	0	0
09	0.5	0	0
10	0.5	0	0
11	0.5	0	0
12	0.5	0	0
13	0.5	0	0
14	0.5	0	0
15	0.5	0	0

FG On/Off OFF Per/5 ns 40 Wid/5 ns 3

Unit Nr. 1

Status

```
FPGA Version: 22505
Timing Resolution: 1.5625E-10 s
Number of Inputs: 16
```

Pattern Settings

Coinc Window [ns] 1.00

Adjust Pattern # 1

Count Time [s] 1.0

Count Time Actual [s] 1.005

Pattern Setting - Click on line to adjust

```
P01: T....X.. ........
P02: X....T.. ........
P03: T....... ........
P04: .....T.. ........
P05: T....T.. ........
P06: X....X.. ........
P07: ........ ........
P08: ........ ........
P09: ........ ........
P10: ........ ........
P11: ........ ........
P12: ........ ........
P13: ........ ........
P14: ........ ........
P15: ........ ........
P16: ........ ........
P17: ........ ........
P18: ........ ........
P19: ........ ........
P20: ........ ........
P21: ........ ........
P22: ........ ........
P23: ........ ........
P24: ........ ........
P25: ........ ........
P26: ........ ........
P27: ........ ........
P28: ........ ........
P29: ........ ........
P30: ........ ........
```

True	1
Off	2
Off	3
Off	4
Off	5
False	6
Off	7
Off	8
Off	9
Off	10
Off	11
Off	12
Off	13
Off	14
Off	15
Off	16

CSV File: Click Me To Set CSV File

Singles

```
S01: 5023149
S02:       0
S03:       0
S04:       0
S05:       0
S06: 5023149
S07:       0
S08:       0
S09:       0
S10:       0
S11:       0
S12:       0
S13:       0
S14:       0
S15:       0
S16:       0
```

Coincidences

```
P01: 5023149
P02: 5023149
P03: 5023149
P04: 5023149
P05:       0
P06:       0
```

Figure 4-IX UQD Logic16 Correlation Viewer

Setup of UQD Logic 16 Correlation Viewer

File location: CD Folder>applications>uqd_apps> UQD_correlation_viewer_V0.31_Bin_> UQD_correlation_counter.exe

Using UQD Correlation Viewer

Parameters

Adjust Input parameters here by changing the following functions.

- ☐ `Tune channel #`
  Enter which channel you would like to adjust (You also do this by clicking the channel in the `Channel Setting` display).
- ☐ `Threshold [V]`
  Enter the threshold level in voltage of your input. Click `All` to apply the change to all inputs.
- ☐ `Delay [ns]`
  Enter amount of time to delay an input signal. Click `All` to apply the change to all inputs.
- ☐ `Polarity`
  Select which edge of the signal you will like to detect, `Positive Edge` (rising) or `Negative Edge` (falling). Click `All` to apply the change to all inputs.

☐ `Configurations`
  - `Revert Default`: Click to reset the patterns to default
  - `Set as Default`: Click set the current configuration of patterns to default
  - `Load From…`: Click to load saved setting of patterns
  - `Save As…` : Click to save current configuration as a file to access later on

☐ `FG On/Off`
Click `ON` or `OFF` to turn on or off the Function Generator. Adjust the `Period/5 ns` and `Wid/5 ns` by entering the desired amount in the designated text box.

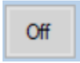
☐ `Connect`
Click to connect to Logic16. If you have multiple units connected to the PC, you can select the unit you want to connect by editing `Unit Nr`.

☐ `Status` display
Shows the status of your device. Any error message will also be shown here.

☐ `Singles` display
Shows the count on each input.

Pattern Settings

Set patterns for coincidence counting using the numbers on the left side. Each number represents an input channel. To set patterns, click the button beside the numbers. You have the following options:



This is the default option. This indicates the device will ignore the associated input.

This indicates the device will count every time there is an active edge on the associated input.

This indicates the device will count every time there is not an active edge on the associated input.

You can set up 64 different patterns. Under `Pattern Settings`, you can view the list of patterns available you can click on each pattern and adjust them individually. Figure # shows examples of a few patterns.

☐ `Coinc Window [ns]`
Enter/adjust number to indicate the preferred coincidence window in ns.

☐ `Adjust Pattern #`
Enter/select the pattern # you want to adjust. You can also do this by clicking the corresponding pattern on the list in `Pattern Settings` display.

☐ `Count time [s]`
Enter count time here to adjust. Since it may sometimes take longer or shorter time, `Actual Count Time` displays the real time for each count. Roughly it will be close to what is indicated by the user.

☐ `Coincidences` display
Shows the counts based on the patterns defined by user.

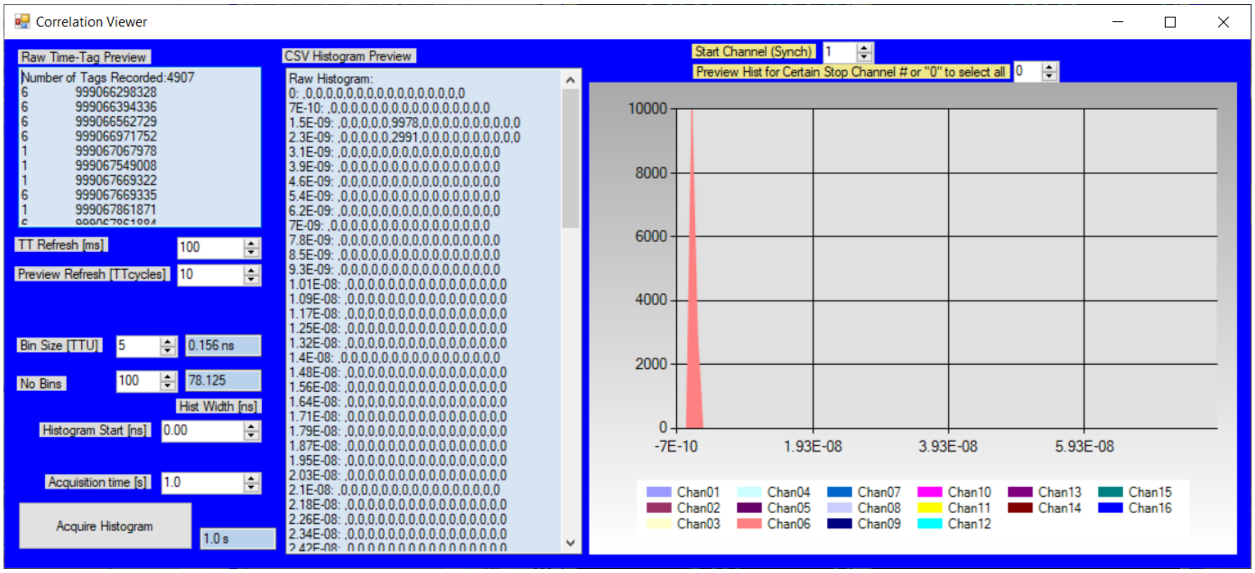
Log Logic data to CSV file

☐ `Click Me To Set CSV File`

Click to choose where to save CSV file and name it. Then, click `Log Logic Data to CSV` and then `Start Saving Data to CSV`. Once you're done collecting your data, click `Saving…Click to Stop.`

Correlation Histogram

Click "Correlation Historgram" button to open correlation histogram window. Here you can acquire a histogram of time tags of your inputs. Histograms settings such as number of bins, bin size, etc. can be adjusted on the right hand side panel. The histogram is display on the left hand side. The raw data is previewed under "CSV Histogram Preview".



Raw Time-Tag Preview

```
Number of Tags Recorded:4907
6          999066298328
6          999066394336
6          999066562729
6          999066971752
1          999067067978
1          999067549008
1          999067669322
6          999067669335
1          999067861871
```

TT Refresh [ms] 100
Preview Refresh [TTcycles] 10
Bin Size [TTU] 5 0.156 ns
No Bins 100 78.125
Hist Width [ns]
Histogram Start [ns] 0.00
Acquisition time [s] 1.0
1.0 s

CSV Histogram Preview

```
Raw Histogram:
0: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
7E-10: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1.5E-09: ,0,0,0,0,0,9978,0,0,0,0,0,0,0,0,0,0
2.3E-09: ,0,0,0,0,0,2991,0,0,0,0,0,0,0,0,0,0
3.1E-09: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3.9E-09: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
4.6E-09: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
5.4E-09: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
6.2E-09: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
7E-09: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
7.8E-09: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8.5E-09: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
9.3E-09: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1.01E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1.09E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1.17E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1.25E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1.32E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1.4E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1.48E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1.56E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1.64E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1.71E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1.79E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1.87E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1.95E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2.03E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2.1E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2.18E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2.26E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2.34E-08: ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Start Channel (Synch) 1
Preview Hist for Certain Stop Channel # or "0" to select all 0

Figure 4-X Correlation Histogram Window

4.3 Application Programming Interface (API)

Logic16 has a sophisticated software interface with built-in multi-processing support. One processor handles USB communication and time-tag preprocessing. The other processor(s) is (are) free for user handling of the read tags (storage / real time calculation).

Logic16 is accessible through a variety of programs. Using the ttInterface.dll (.Net) file provided in the CD, users can program Logic16 in C#, LabVIEW, Visual Basics, MATLAB and Python. C++ is also accessible with headers provided.

Disclaimer: Before accessing the .dll file on an API, ensure it is unblocked. To do this right-click on the .dll file, go to "Properties" and under "General", check off "Unblock". Otherwise, Windows will prevent the library to load.

4.3.1 .NET Setup

.NET assemblies can be integrated in most programming environments, including LabVIEW and C#. The software interface is contained in the class ttInterface that is present in the file ttInterface.dll. The .dll file can be found in the CD driver provided under the folder named “TimeTagExplorer” within applications. This is only available for Windows.

LabVIEW

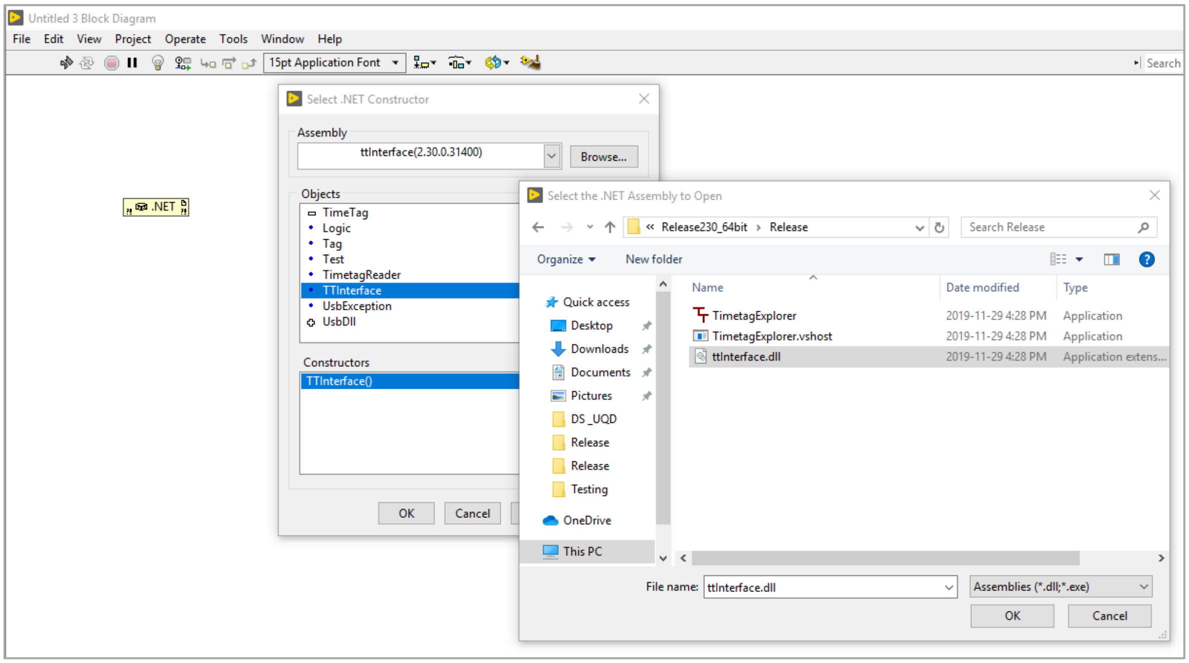
Once a new Labview project is opened, create a contruct node to access the ttInterface.dll. You can do so by clicking on the node and browse to file location. Then create and connect an invoke node to access all objects and methods within the file.

Note: ttInferface.dll uses .NET framework 4.0 which is not officially supported by older versions of LabVIEW. But it still works because none of the 4.0 features is actually used.

To use 4.0, the following configuration file is required:

The configuration file must be placed next to `LabVIEW.exe` and must be named `LabVIEW.exe.config`. The following example instructs LabVIEW to load the CLR 4.0:

```
<configuration>
<startup useLegacyV2RuntimeActivationPolicy="true">
<supportedRuntime version="v4.0.30319"/>
</startup>
</configuration>
```



Assembly: ttInterface(2.30.0.31400)

File name: ttInterface.dll — Assemblies (*.dll;*.exe)

Figure 4-XI LabVIEW: Accessing ttInterface.dll through a constructor node.

Visual Studio (C# and Visual Basic)

Once you create a new project, access ttInterface.dll by adding reference (under "Project" tab). Browse to select ttInterface.dll from the CD driver.

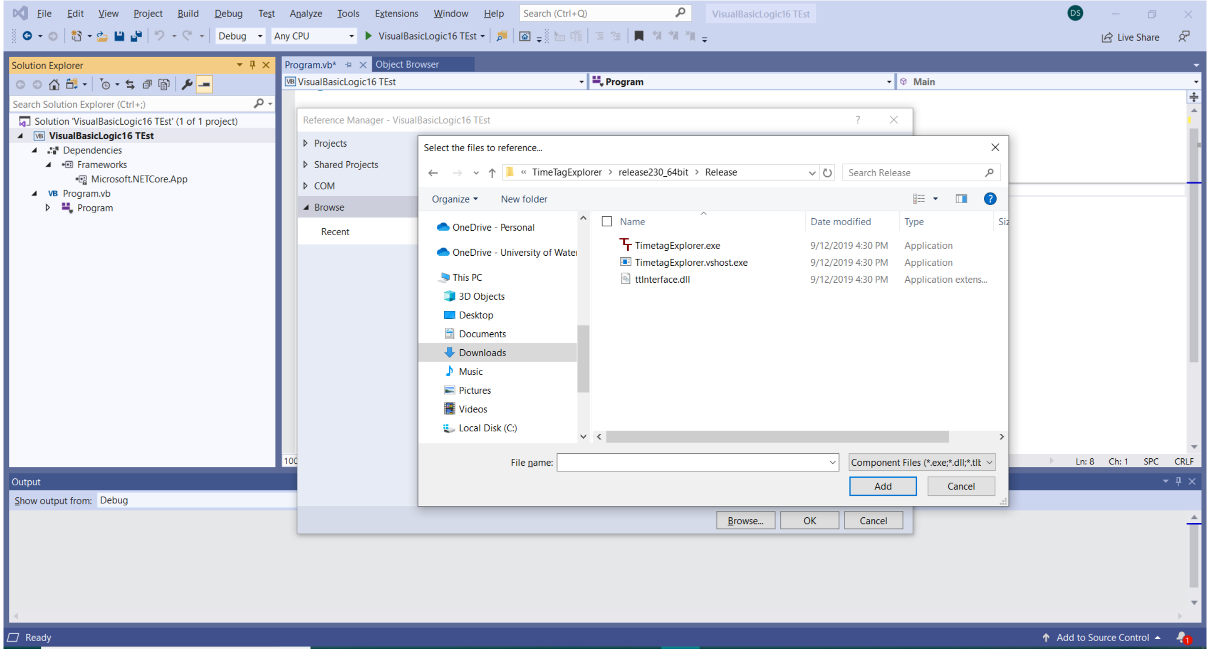

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
VisualBasicLogic16 TEst
Debug
Any CPU
Live Share
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'VisualBasicLogic16 TEst' (1 of 1 project)
VisualBasicLogic16 TEst
Dependencies
Frameworks
Microsoft.NETCore.App
Program.vb
Program
Program.vb*
Object Browser
Program
Main
Reference Manager - VisualBasicLogic16 TEst
Projects
Shared Projects
COM
Browse
Recent
Select the files to reference...
TimeTagExplorer › release230_64bit › Release
Search Release
Organize
New folder
OneDrive - Personal
OneDrive - University of Wate
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Name
Date modified
Type
TimetagExplorer.exe 9/12/2019 4:30 PM Application
TimetagExplorer.vshost.exe 9/12/2019 4:30 PM Application
ttInterface.dll 9/12/2019 4:30 PM Application extens...
File name:
Component Files (*.exe;*.dll;*.tlt
Add
Cancel
Browse...
OK
Cancel
Output
Show output from: Debug
Ln: 8 Ch: 1 SPC CRLF
Ready
Add to Source Control

Figure 4-XII Visual Studio: Add ttInterface.dll as a reference in order access all objects and methods

Simple C# Demo Source Code

The demo source code is a very simple C# project that shows the basic usage of the time tag interface. The project file is “SimpleTimetagDemo.sln”. It can be opened with Microsoft Visual Studio 2010. The “Express” version of Visual Studio will also work.

This simple program fragment shows how to use the interface:

```
ttInterface = new TTInterface(); //Create object
ttInterface.Open(1);            //Open device No 1

ttInterface.Calibrate();

ttInterface.StartTimetags();     //Start measurement

byte[] channels;

long[] times;

int count = ttInterface.ReadTags(channels, times);

ttInterface.StopTimetags();

ttInterface.Close();
```

On error conditions, an exception of type `TimeTag.UsbException` is thrown.

Python

In Python, the .NET functions are accessible using the Pythonnet package. UQD_Logic_Python_Frequency_Analzyer.py is a simple example that shows how to use the UQD logic, both in Logic counting mode, as well as time-tagging mode.

Note: For speed and performance consider using other platforms such as C# or VB.

To access the ttInterface.dll file in python, include the following into your code. Modify the file path as necessary.

```
import time
import sys

#please change paths to the location of ttInterface.dll

sys.path.append("C:\\Users\\Work\\Documents\\UQD\\TTexplorers\\TT
Explorer_2_28_32bit")

import clr
clr.AddReference("C:\\Users\\Work\\Documents\\UQD\\TTexplorers\\T
TExplorer_2_28_32bit\\ttInterface")

from System import Array, Byte, Int16, Int32, Int64, UInt32, UInt16

from TimeTag import TTInterface, Logic, UsbException
```

4.3.2 C++ Setup

A C++ interface is available for Windows as well as Linux. The Windows interface is compiles using Microsoft Visual Studio 2013. The files are contained in the folder "CTimeTag".

File	Description
include\CTimeTag.h	General header file
include\CLogic.h	Header file for Logic special functions
Win32\CTimeTagLib.lib Win32\CTimeTagLibDebug.lib	Windows 32 bit
Win64\CTimeTagLib.lib Win64\CTimeTagLibDebug.lib	Windows 64 bit
Linux\libtimetag32.so	Linux 32 bit
Linux\libtimetag64.so	Linux 64 bit

Simple C++ Program

```
#include <stdio.h>
#include <stdlib.h>
#include "../CTimeTag/Include/CTimeTag.h"
#include "../CTimeTag/Include/CLogic.h"

using namespace TimeTag;
CTimeTag timetag;

timetag.Open(1);                //Open device No 1

timetag.Calibrate();

timetag.StartTimetags();     //Start measurement

      unsigned char *chan;
      long long *time;
      int count= timetag.ReadTags(chan, time);
timetag.StopTimetags();

timetag.Close();
```

On error conditions, an exception of type `TimeTag::Exception is thrown.`

How to use the C++ Demo Source Code

To use the demo program, just copy the two Folders “CppDemo” and “CTimeTag” into the same parent folder.

Windows:

On Windows, open CppDemo.sln using Visual Studio 2010 and compile.

Linux:

On Linux, copy libtimetag32.so or libtimetag64.so to your lib directory. If you have not already done so, get the g++ compiler and libusb development version.

```
sudo apt-get install g++-4.6-multilib

sudo apt-get install libusb-1.0-0-dev
```

Then navigate to the CppDemo folder and type “make CppDemo32” or “make CppDemo64”

To run the program, you need administrator privileges.

4.3.3 API Methods

Objects

`TTInterface()`

This is for programs using ttInterface.dll (.NET). TTInterface() must be initialised in order to access all functions required to operate the device.

Syntax:

C#	`myTagger = new TTInterface();`
VB	`myTagger = New TTInterface()`
Python	`myTagger = TTInterface()`

`Logic (myTagger)`

This is for programs using ttInterface.dll (.NET). The constructor takes a `ttInterface` object which must be already created, and creates a `Logic` object. The `ttInterface` object can already be opened or not. (Logic Mode)

Syntax:

C#	`myLogic = new Logic(myTagger);`
VB	`myLogic = New Logic(myTagger)`
Python	`myLogic = Logic(myTagger)`

`CTimeTag`

This class is only for C++, which must be initialised in order to access all functions required to operate the device.

Syntax:

C++	`CTimeTag timetag;`

`CLogic`

This class is only for C++, which must be initialised in order to access all functions required for correlation counting (Logic Mode).

Syntax:

C++	`CLogic logic(timetag);`

Basic Functions

`Open(Number)`

This function connects to the device. It has to be called before any other function is called.

`Number`	`>=1`	The parameter is used only when more than one units are connected at the same time. If only one unit is used, this value must be set to 1

Syntax:

C# `myTagger.Open(int Number);`

C++ `timetag.Open(int Number);`

`Calibrate()`

The calibration function increases the accuracy of the device. It needs 4-10 seconds to execute. When the calibration is used, it should be the first function to be called after the “Open” function. Note: Intrinsic delays of each input may vary after the calibration procedure is performed.

Syntax:

C# `myTagger.Calibrate();`

C++ `timetag.Calibrate();`

`Close()`

This function should be called before the program terminates.

Syntax:

C# `myTagger.Close();`

C++ `timetag.Close();`

`SetLedBrightness(int percent)`

The Brightness of the LED on the front panel can be changed.

Syntax:

C# `myTagger.SetLedBrightness(int percent)`

C++ `timetag.SetLedBrightness(int percent)`

`GetFpgaVersion()`

This function returns the current version of the FPGA design. It is used for debugging purposes only.

Syntax:

C# `myTagger.GetFpgaVersion()`

C++ `timetag.GetFpgaVersion()`

`GetResolution()`

This function returns the time resolution of the device. It should be used to calculate absolute time values. The function returns either 78.125E-12 or 156.25E-12 seconds.

Syntax:

C#	`myTagger.GetResolution()`
C++	`timetag.GetResolution()`

`GetNoInputs()`

This function returns the number of inputs installed on the device. It is used for debugging purposes only.

Syntax:

C#	`myTagger.GetNoInputs()`
C++	`timetag.GetNoInputs()`

Setup Functions

`SetInputThreshold(int input, double voltage)`

Input	[1..16]	Number of the input to change.
Voltage	[-2.0..2.0]	Threshold voltage in volt

Syntax:

C# `myTagger.SetInputThreshold(int input, double voltage)`

C++ `timetag.SetInputThreshold(int input, double voltage)`

`SetInversionMask(int mask)`

The inputs are edge sensitive. The positive edge is used as standard. With this function the relevant edge can be changed to the negative edge. The mask is coded binary. When the corresponding bit is high, the negative edge is used. The corresponding bits are bits 16 to 1. Bit 0 is unused.

Syntax:

C# `myTagger.SetInversionMask(int mask)`

C++ `timetag.SetInversionMask(int mask)`

`SetDelay(int input, int delay)`

All input signals can be delayed internally. This is useful to compensate external cable delays.

`Input`	`[1..16]`	Number of the input to change.
`Delay`	`18 bit value`	Delay in internal units. (See GetResolution())

Syntax:

C# `myTagger.SetDelay(int input, int delay)`

C++ `timetag.SetDelay(int input, int delay)`

`SetFG(int period, int high)`

This function defines the rectangular signal on connector output 4. Both values are defined in units of 5 ns. The maximum width of both values is 28 bit which gives a maximum time of about 1.3 sec. Example: SetFG (20, 10) gives 10 MHz and 50 % duty cycle.

Syntax:

C# `myTagger.SetFG(int period, int high)`

C++ `timetag.SetFG(int period, int high)`

`Use10MHz(Boolean use)`

When the 10 MHz input is switched on, but no valid signal is connected to the input, an error flag is set and the error led on the front panel is lit.

Note: When using the 10 MHz reference, input 16 on the 156.25 ps devices and input 8 on the 78.125 ps devices cannot be used.

Syntax:

C#	`myTagger.Use10MHz(bool use)`
C++	`timetag.Use10MHz(bool use)`

`FreezeSingleCounter()`

This function stores all the single counters synchronously. This function also returns the time between the last two calls to FreezeSingleCounter. The time is expressed in 5 ns ticks.

Syntax:

C#	`myTagger.FreezeSingleCounter()`
C++	`timetag.FreezeSingleCounter()`

`GetSingleCount(int input)`

This returns the number of input pulses in between the last two calls of `FreezeSingleCounter`.

This example calculates the frequency of input 4:

```
int time = FreezeSingleCounter();

int pulses = GetSingleCount(4);

double frequency = 200e6 * pulses / time;
```

Syntax:

C#	`myTagger.GetSingleCount(int input)`
C++	`timetag.GetSingleCount(int input)`

TimeTag Mode

In this mode, the internal ram is used as a fifo. The tags are transmitted to the computer by a background thread. With this configuration, very high transmission rates can be achieved. At an internal fifo overflow, the `DataOverflow` flag is set and the error led on the front panel is lit.

`StartTimetags()`

This function puts the device into time-tag readout mode. The time tags are written into the internal fifo. The background thread starts to read the time tags into the RAM of the computer.

Syntax:

C#	`myTagger.StartTimetags()`
C++	`timetag.StartTimetags()`

`StopTimetags()`

The tags are no longer written into internal fifo. The background thread is stopped.

Syntax:

C#	`myTagger.StopTimetags()`
C++	`timetag.StopTimetags()`

`ReadTags(channel, times)`

This returns value: size of arrays. Time[n] is the absolute time in internal units (78.125 or 156.25 ps). Channel [n] is the number of the corresponding input. The first input is number 1. The array is sorted by time. When two tags occur at the same time, then the tag with the smaller channel number is transmitted first. The arrays can be used until the next call of ReadTags. When the data is needed for a longer time it has to be copied to a different array. The array is allocated by the driver software and returned to user code with an out parameter. Returning with an out parameter is similar to returning values by a pointer or reference in C / C++;

The “virtual channel” 30 is used for the overflow tag. The overflow tag is sent whenever a data overflow error occurs. The data overflow tag indicates, that some data is missing at this point.

When there are no tags present, this function may wait up to 300 ms before it returns a zero result. When you don’t want this behavior, use TagsPresent() to check availability first.

Syntax:

C#	`myTagger.ReadTags(out byte[] channels, out long[] times);`
C++	`timetag.ReadTags(unsigned char * channels, long long *times)`

`TagsPresent()`

This function returns true when tags are ready to read. This function is optional.

Syntax:

C#	`myTagger.TagsPresent()`
C++	`timetag.TagsPresent()`

Time Tag Filter

When the time-tag filter is on, the time-tags are transmitted only, when they appear in groups. This feature helps to reduce the USB bandwidth and PC load. The filter is able to process rates of up to 190 MHz.

`SetFilterMinCount(int MinCount)`

This function defines the minimum size of a group to be transmitted. MinCount can be set between 1 and 10 counts. Setting MinCount to 1 switches the filter off, all tags are transmitted.

Syntax:

C#	`myTagger.SetFilterMinCount (int MinCount)`
C++	`timetag.SetFilterMinCount (int MinCount)`

`SetFilterMaxTime(int MaxTime)`

MaxTime defines the maximum time between two pulses in the same group. When the time between two pulses is bigger than “MaxTime”, the two pulses are considered to be in different groups. MaxTime is given in internal units.

Example: When FilterMinCount is 10 and FilterMaxTime is 1 µs, then the maximum possible group size would be 9 µs.

Syntax:

C#	`myTagger.SetFilterMaxTime (int MaxTime)`
C++	`timetag.SetFilterMaxTime (int MaxTime)`

`SetFilterException(int exception)`

Some inputs can be excluded from the filter. Excluded inputs are always transmitted. They do not participate in groups. The filter exceptions are bit-coded. To exclude Input n from the filter, set the bit n-1 in the exception mask.

Examples:

0	No exceptions
1	Input 1 excluded
2	Input 2 excluded
3	Inputs 1 and 2 excluded
4	Input 3 excluded

Syntax:

C# `myTagger.SetFilterExceptions (int Exceptions)`

C++ `myTagger.SetFilterExceptions (int Exceptions)`

Level Gate

When the voltage on the gate Input is higher than the threshold voltage, then the device operates normally: The time tags of all inputs are stored in internal RAM and transmitted via USB. When the voltage is below the threshold voltage, then the input signals are ignored and no tags are stored in internal RAM. The threshold voltage is the voltage of channel 9.

This feature is not present in all devices. This input has jitter and timing resolution of 5 ns.

`UseLevelGate(bool)`

False: Normal operation, the gate input will be ignored.

True: Level Gate Operation, tags are stored only when the gate input is high.

Syntax:

C# `myTagger.UseLevelGate (bool)`

C++ `timetag.UseLevelGate (bool)`

`LevelGateActive()`

Returns true, when the gate input is above the input threshold

Syntax:

C# `myTagger.LevelGateActive()`

C++ `timetag.LevelGateActive()`

Edge Gate

This gating offers fine grained timing control and very low jitter. The width of the gating window can be adjusted in steps of the internal resolution. The position of the gate can be adjusted too. This offers very flexible control of the gate.

The gate is opened a fixed time after the active edge of input 8. This fixed time can be set with `SetDelay(8, delay);`

The rising edge is the standard active edge. This can be change by `SetInversionMask();`

A negative gate delay is possible too. To achieve this, the delay of input 8 must be set to 0 and the delay of all other inputs must be set to the magnitude of the desired delay value. The gate is open for a fixed time interval. This interval can be adjusted in in internal units.

`UseTimetagGate(bool use)`

This function switches edge sensitive gating on or off.

Syntax:

C# `myTagger.UseTimetagGate(bool use)`

C++ `timetag.UseTimetagGate(bool use)`

`SetGateWidth(int duration)`

This function sets the width of the gate. The parameter duration is given in internal units.

Syntax:

C#	`myTagger.SetGateWidth(int duration)`
C++	`timetag.SetGateWidth(int duration)`

Save to file

The software interface offers a high performance interface to save data to disk. The data is saved in a compressed, binary format and can be converted to ASCII offline. This way the full rate of 11 MHz can be saved to disk on a modern computer.

It is best explained by an example:

Saving:

```
TTInterface tti= new TTInterface();

tti.Open()

TimeTagReader r= tti.GetReader();

r.StartSaving(“rawdatafile.tt”);

//Wait some time

//Monitor r.SavedTags when you like

r.StopSaving();

tti.Close();
```

Converting

```
TTInterface tti= new TTInterface();

TimeTagReader r= tti.GetReader();

r.StartConverting(“rawdatafile.tt”, “tags.txt”);

r.WaitUntilConversionFinished();
```

For converting you don’t have to call Open() and you don’t even have to have a unit connected.

Of course you can do the conversion right after StopSaving().

Conversion Mode

The generated file can be either text or binary. Binary mode is selected by r.`SetConversionBinary` (`true`). The default is text mode.

The interface functions are located in the dll ttInterface.dll. There is also support for C++, both for Windows and Linux environments.

Logic mode

`SwitchLogicMode()`

This method must be called before the other logic functions can be used.

Syntax:

C#	`myLogic.SwitchLogicMode()`
C++	`logic.SwitchLogicMode()`

`SetWindowWidth(int window)`

This method sets width of the coincidence window. The parameter window is given in internal units (156.25 ps or 78.125 ps). The window width can be set as high as 2^{24}-1.

Syntax:

C#	`myLogic.SetWindowWidth(int window)`
C++	`logic.SetWindowWidth(int window)`

`ReadLogic()`

This method reads the counter out of the device. The return parameter is not needed (used for debug purposes only). The data is automatically stored in the Logic object for later processing. This method can be called any time after calling `SwitchLogicMode()`. The time between calls to `ReadLogic()` defines the measurement time interval of the captured data.

Note: The device uses a double buffered memory system. The new measurement starts immediately and is running in the background during the call to ReadLogic(). For this reason not a single pulse is omitted during readout.

Syntax:

C#	`myLogic.ReadLogic()`
C++	`logic.ReadLogic()`

`CalcCount(int pos, int neg)`

This method operates on the data read by the last call to ReadLogic(). It calculates how often a certain event pattern has occurred in the last measurement interval. The parameters `pos` and `neg` are bit coded. The rightmost bit corresponds to input 1, the next to input 2, and so on.

The parameter `pos` is given by the integer value of the binary number defined by the bit coded Coincidence Event. It defines the inputs that must have an active edge in the coincidence window for the event to be counted, with range [0..65535].

The parameter `neg` defines the inputs that must have no active edge in the coincidence window for the pattern to be counted. The parameter neg is optional, with range [0..65535].

Syntax:

C#	`myLogic.CalcCountPos(int pos)`
C++	`logic.CalcCountPos(int pos)`

`pos`	**`neg`**	**`pos` Coincidence Pattern**	**Coincidence Event**
1	0	0000000000000001	Singles of input 1
2	0	0000000000000010	Singles of input 2
4	0	0000000000000100	Singles of input 3
8	0	0000000000001000	Singles of input 4
3	0	0000000000000011	Coincidence of input 1 and 2
7	0	0000000000000111	Coincidence of inputs 1, 2, and 3
3	4	0000000000000011	Coincidence of inputs 1 and 2 without interaction of input 3

`CalcCountPos(int pos)`

This method equals CalcCount except that the neg parameter is always 0. It has better run time.

Syntax:

C#	`myLogic.CalcCountPos(int pos)`
C++	`logic.CalcCountPos(int pos)`

`GetTimeCounter()`

The time counter measures the time between the last two calls to ReadLogic. The result is given in multiples of 5 ns. The time when ReadLogic() is called will vary over time because of the limited realtime performance of personal computers. The correct count rates can be obtained when the count values from CalcCount are divided by the result from GetTimeCounter().

Syntax:

C#	`myLogic.GetTimeCounter()`
C++	`logic.GetTimeCounter()`

Outputs

`SetOutputEventCount(int events)`

This function is an advanced feature that can be used to fine tune the delay of the outputs. Roughly speaking, events is the number of events you expect to occur in one 5 ns time slice under worst-case conditions.

Increasing the value by one will increase the delay of the outputs by 10 ns. With the standard setting of 5 the output delay will be 350 ns.

When the input rate is too high for the given event count, the OutTooLate Error Flag will be raised. No output pulse will be generated in this condition. There is never an output pulse generated in the wrong timing. For this reason OutTooLate is more a kind of warning, not a hard error condition. It just means that not all pulses are generated.

Hint: When two events occur at the exact same time, the event coming from the input with the smaller input number will be processed first. For this reason input 1 has a slightly better real time performance than input 16.

Syntax:

```
C#      myLogic.SetOutputEventCount(int events)
C++     logic.SetOutputEventCount(int events)
```

`SetOutputPattern(int output, int pos, int neg)`

This function sets the pattern of the output pulses based on the input coincidence events. The parameter output identifies which output to change, with range [1..3].

The parameter pos is the bit coded value of the Coincidence Event of the inputs that must be present, with range [0..65535] as for CalcCount.

The parameter neg is the bit coded value of the Coincidence Event of the inputs that must not be present, with range [0..65535].

Method Call	**Result**
`SetOutputPattern(1, 1, 0)`	Output 1 fires when input 1 is present
`SetOutputPattern(2, 6, 0)`	Output 2 fires when inputs 2 and 3 are present within the coincidence window
`SetOutputPattern(3, 6, 8)`	Output 3 fires when inputs 2 and 3 are present and input 4 is not present at the same time

Syntax:

```
C#      myLogic.SetOutputPattern(int output, int pos, int neg)
C++     logic.SetOutputPattern(int output, int pos, int neg)
```

`SetOutputWidth(int width)`

This function defines the length of the generated output pulses. The parameter is given in 5 ns increments. The maximum value is 255.

Syntax:

```
C#      myLogic.SetOutputWidth(int width)
C++     logic.SetOutputWidth(int width)
```

Errors

`ReadErrorFlags()`

This function returns the internal error flags.Calling this function clears the flags in the device.

Syntax:

C# `myTagger.ReadErrorFlags()`

C++ `timetag.ReadErrorFlags()`

`GetErrorText (int flags)`

This function translates the error flags to a short text that can be displayed on the user interface.

Syntax:

C# `myTagger.GetErrorText(int flags)`

C++ `Timetag.GetErrorText(int flags)`

Flag No	**Mask Value**	**Flag Name**	
0	1	`DataOverflow`	An overflow in the 512 k values SRAM FIFO has been detected. The time-tag generation rate is higher than the USB transmission rate.
1	2	`NegFifoOverflow`	Internal reason, should never occur
2	4	`PosFifoOverflow`	Internal reason, should never occur
3	8	`DoubleError`	One input had two pulses within the coincidence window.
4	16	`InputFifoOverflow`	More than 1024 successive tags were detected with a rate greater 100 MHz
5	32	`10MHzHardError`	The 10 MHz input is not connected or connected to a wrong type of signal.
6	64	`10MHzSoftError`	The 10 MHz input is connected, but the frequency is not 10 MHz.
7	128	`OutFifoOverflow`	Internal error, should never occur
8	256	`OutDoublePulse`	An output pulse was generated, while another pulse was still present on the same output. The pulse length is too long for the given rate.
9	512	`OutTooLate`	The internal processing was too slow. This is because the output event queue is too small for the given rate. Increase the value with `SetOutputEventQueue()`
28		`OutOfSequence`	This is an internal error generated in the PC software. It is generated when the tags are not sorted correctly. Plase contact factory.

4.3.4 **Sample Codes**

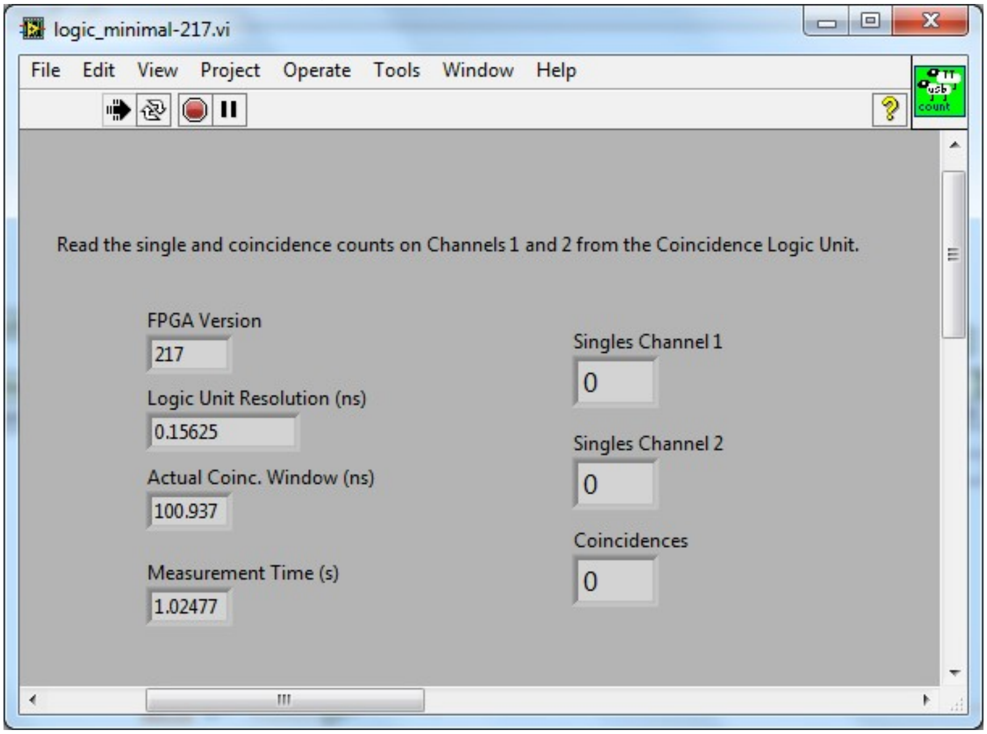
Visual Basic

Minimal implementations of Logic functions

These implementations consist of a small but useful script that opens the Coincidence Logic Unit, sets some parameters, and reads once the single and coincidence counts on channels one and two.

```
Imports TimeTag


Public Class Form1


    Dim myTagger As TTInterface
    Dim mylogic As Logic
    Dim Resolution As Double
    Dim Errors As Integer



    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        Dim Singles1, Singles2, Coincidences As Integer
        Dim FPGA_v, TimerCounter As Double

        'open connection to device
        myTagger = New TTInterface
        myTagger.Open()

        'get some basic info and display
        Resolution = myTagger.GetResolution()
        FPGA_v = myTagger.GetFpgaVersion()
        MsgBox("FPGA Version: " & FPGA_v.ToString & vbCrLf & _
               "Timing Resolution: " & Resolution.ToString & " s" &
               vbCrLf & vbCrLf & vbCrLf & _
               vbTab & vbTab & "Start Measurment Now?")

        'set input thresholds for channel 1 and 2
        myTagger.SetInputThreshold(1, 0.3)
        myTagger.SetInputThreshold(2, 0.3)

        'initiate logic mode
        mylogic = New
        Logic(myTagger)
        mylogic.SwitchLogicMode()

        'conicidence window (in units of 156ps)
        mylogic.SetWindowWidth(20)
        'count events for  1 sec:
        'take a first snapshot of counters
        mylogic.ReadLogic()

        'delay thread for about 1sec
        Threading.Thread.Sleep(1000)

        'take second snapshot of counters
        mylogic.ReadLogic()

        'how long was the true measurement time between the two
              ReadLogic comands? Given in multiples of 5ns,
              therfore we multiply by this unit to obtain sec
              TimerCounter = mylogic.GetTimeCounter() * 0.000000005
              'retrive observed counts for singles 1,2 and
              coincidences 12
              singles1 = mylogic.CalcCountPos(1)
              singles2 = mylogic.CalcCountPos(2)
              coincidences = mylogic.CalcCountPos(3)

        'show  data
        MsgBox("Measurment Time: " & vbTab & vbTab &
TimerCounter.ToString("F4") & " s" & vbCrLf & _
               "Singles Channel 1:" & vbTab & vbTab &
               Singles1.ToString & vbCrLf & _
               "Singles Channel 2:" & vbTab & vbTab &
              Singles2.ToString & vbCrLf & _
               "Coincidences Chs 1&2:  " & vbTab & Coincidences.ToString)
        Close()

    End Sub


    Private Sub Form1_FormClosing(sender As Object, e As
FormClosingEventArgs) Handles Me.FormClosing
        MsgBox("Thanks for testing the counter. Bye")

        ' close connection to device
        myTagger.Close()

    End Sub
      End Class
```

LabVIEW Sample Program



Read the single and coincidence counts on Channels 1 and 2 from the Coincidence Logic Unit.

FPGA Version: 217

Logic Unit Resolution (ns): 0.15625

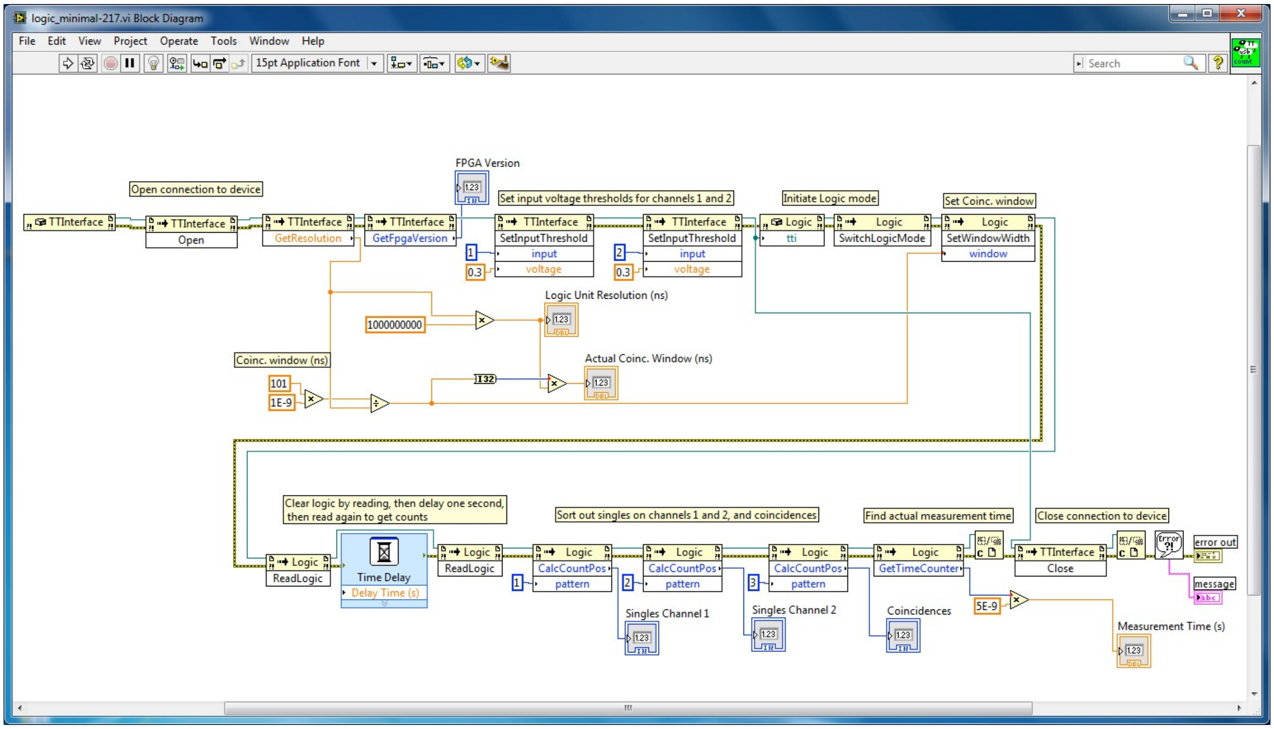
Actual Coinc. Window (ns): 100.937

Measurement Time (s): 1.02477

Singles Channel 1: 0

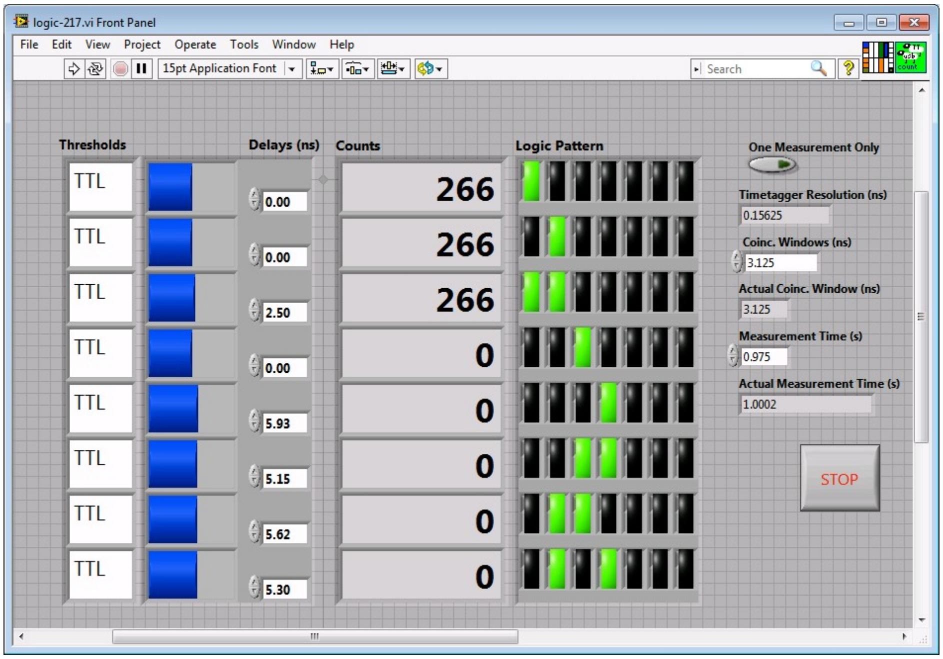
Singles Channel 2: 0

Coincidences: 0



Multi-Channel LabView Implementation of logic functions

This description is based on the LabView VI logic-*ver*.vi (current *ver* is 217) and sub-VI generate logic pattern2.vi. Below is the standard front panel.



Thresholds	Delays (ns)	Counts	Logic Pattern
TTL	0.00	266	🟩⬛⬛⬛⬛⬛⬛
TTL	0.00	266	⬛🟩⬛⬛⬛⬛⬛
TTL	2.50	266	🟩🟩⬛⬛⬛⬛⬛
TTL	0.00	0	⬛⬛🟩⬛⬛⬛⬛
TTL	5.93	0	⬛⬛⬛🟩⬛⬛⬛
TTL	5.15	0	⬛⬛🟩🟩⬛⬛⬛
TTL	5.62	0	⬛🟩🟩⬛⬛⬛⬛
TTL	5.30	0	⬛🟩⬛🟩⬛⬛⬛

One Measurement Only

Timetagger Resolution (ns) 0.15625

Coinc. Windows (ns) 3.125

Actual Coinc. Window (ns) 3.125

Measurement Time (s) 0.975

Actual Measurement Time (s) 1.0002

Thresholds
Thresholds allows the voltage threshold to be set for each channel. Two standard voltages are NIM and TTL.

Delays
Delays allows the delay for each channel in nanoseconds to be set, in order to align coincident counts between channels. The delays can only be set to multiples of the Coincidence Logic Unit's resolution.

Counts
Counts displays the number of events in the last Measurement Time corresponding to a given logic pattern.

Logic Pattern
Logic Pattern is the pattern of coincidences to report in counts. Channel 1 is the leftmost button and channel 6 is the rightmost (expand the control to see more channels). Green means an event on the channel must be present, red means an event on the channel must not be present, and black means to ignore the channel. For example, the first three patterns in the above screenshot are for single counts on channel 1, single counts on channel 2, and coincident counts between channels 1 and 2, respectively.

STOP
Always use the STOP button, rather than LabView's Abort Execution so that the connection to the Coincidence Logic Unit is closed correctly.

Other settings

One Measurement Only, when enabled, returns the counts for only one Measurement Time, and then stops execution (otherwise measurements run continuously). This is useful, for example, when changing settings between each measurement.

Time tagger Resolution reports the resolution of the Coincidence Logic Unit in nanoseconds.

Coinc. Window controls the time window in nanoseconds within which counts on different channels are counted as coincident.

Actual Coinc. Window reports the coincidence window actually used in nanoseconds, which must be a multiple of the Coincidence Logic Unit's resolution.

Measurement Time chooses the time in seconds between reading the Coincidence Logic Unit's buffer, meaning Counts are displayed as summed over the Measurement Time.

Actual Measurement Time reports the real time in seconds between reading the Counts, due to finite processing time of the PC.

Section 5

FAQ

What additional equipment do I need to connect my single photon detector to Logic16?

Nothing! (Just an SMA connector)

Standard single photon detectors output TTL pulses and these are easily read by the Logic16 through its SMA inputs (no need for a TTL-to-NIM converter).

- To be registered as an event on Logic16, the TTL pulses must endure for at least 300 ps above the (user-set) discriminator threshold. The inputs have a discriminator range of +/- 2.0V, but can tolerate a signal range of more than +/- 3.0V without issue.
- For standard (gated) infrared photon detectors, consider using the integrated pulse generator and coincidence-triggered outputs to control gating!

How can I switch between the Logic16 Configuration A (156 ps, 16-channel) and B (78 ps, 8-channel)?

Switching configurations requires updating the firmware of the FPGA that is housed within the Logic16 unit. This can be done by the user in-lab, but not during a measurement.

How can I record time-tags over many hours in my multi-photon experiment, without accumulating excessively large data files?

Use the time-tag filter. By transferring data only after a set number of tags accumulate within a given time window, one can dramatically reduce the amount of data being transferred to the host PC. The photon-triplet experiments in Phys. Rev. Lett. 118, 153602 (2017) – arxiv:1609.07508 – stored filtered time-tag data over 20 h, using Logic16 units in time-tagging mode.

If the maximum transfer rate is 10 MHz, how can I use the logic unit for time-resolved measurements with a higher rate pulsed laser?

Using the time-tag filter with a filter width just less than the period of the laser, Logic16 will only transfer those time tags for which a detection occurred. One can use, e.g. a 80 MHz Ti-Sapphire laser, and only transfer events of interest (detection events + corresponding laser signals).

What if I don't need ALL the possible correlation patterns in coincidence counting mode? Can I select specific ones?

Yes. Logic16 pre-sifts all possible coincidences that occurred within a given time window and transfers the meta data to the PC. The PC will receive information about all possible correlations and the driver will extract only coincidences requested by the user.

What software is available? Can it be customized?

- Time Tag Explorer – test installation and operation of the Logic16. This is to be used for initial set-up trouble-shooting, but does not provide data acquisition (saving) capability.
- UQD Logic Counter – basic coincidence counting and data recording
- Customization – UQDevices suggests that experimentalists develop a customized data acquisition software in their preferred DAQ environment (LabView, C#, Matlab, etc.). If you have questions, requests or feedback regarding available software please contact us.

Does the Logic16 unit come with a power supply?

No. The unit has an input port for a DC power supply (12V/≥1.5 A) or can be powered by NIM crate (+ 12V/1A , -12V/0.5A). An example of a compatible DC power supply is here.

Can two or more Logic16 units be used together to increase the number of channels

Yes. One can use n x Logic16 units to effectively obtain n*15 channels.

This is accomplished by syncing the units using a time reference signal. To do so, split a time reference signal (e.g. 1 kHz signal from a function generator) and input this reference into one channel on each unit. For example, two Logic16 units were used to employ 30 channels to time-tag a SPAD array in the following experiments:

– Opt. Express 24, 20947-20955 (2016), arxiv.org:1606.04110

–J. Appl. Phys. 116, 143101 (2014), arxiv:1406.4245

Section 6

Revision History

Date	**Document Version**	**FPGA Version**	**Dll Version**	**Remark**
2019-12-20	3.0	2.30	2.30	User Manual Version 3.0 is a combination of Time Tag User Manual 2_3 and Logic User Manual 1_5_2.